

Soft Operators Decision Trees

Uncertainty and stability related issues

Eva Barrena Algara

Vom Fachbereich Mathematik
der Technischen Universität Kaiserslautern
zur Verleihung des Akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte Dissertation.

1. Gutachter: Prof. Dr. Dieter Prätzel-Wolters

2. Gutachter: Prof. Dr. Stefan Nickel

Vollzug der Promotion: 27.08.2007

To my parents and my sister.

Acknowledgments

I would like to thank all the people who helped and supported me in the process of writing this thesis.

First of all, I would like to express my sincere gratitude to my supervisors, Prof. Dr. Dieter Prätzel-Wolters and Dr. Patrick Lang, for providing me with this opportunity, as well as for having taken interest in my work and spending time to read and suggest improvements in the manuscript. Also I would like to extend my sincere thanks to the department of Adaptive Systems of the Fraunhofer "Institut für Techno- und Wirtschaftsmathematik" (ITWM) which funded this work. At this point I would like to thank Dr. Alexander Dreyer for his helpful corrections and Jan Hauth for his contagious enthusiasm for mathematics and for always being available for discussions and help about my thesis. Next, I owe very big thanks to Dr. Beatriz Clavero, Isabel Rojo and Teresa Jiménez for the interesting discussions and precious contributions.

My special thanks go to my family for their love and care all along my life and for being so near in spite of the distance during the period of my thesis.

Further, I owe a heartfelt thank to Markus for being the best support I could ever wish.

Last but not the least, thanks to my friends for helping me in many different ways.

Contents

1	Introduction	6
1.1	Outline of this work	8
2	Topology of the SODT	11
2.1	Introduction	11
2.2	Basics of graph theory	13
2.3	The degree of a vertex	16
2.4	Paths and cycles	16
2.5	Connectivity	17
2.6	Digraphs	18
2.7	Trees and forests	19
2.7.1	Roots and orderings	21
2.7.2	Partition of a tree into levels	22
2.7.3	Theorem for labeling binary rooted trees	24
2.7.4	Other notions of trees	25
3	Decision trees	28
3.1	Introduction	28
3.2	Outline of the chapter	29
3.3	Notation and definitions	30
3.3.1	Types of variables	31
3.3.2	Classifiers as partitions	32
3.4	Binary tree structured classifiers	32
3.4.1	How does split decomposition work?	35
3.4.2	Formal definition of Decision Trees	38
3.5	DT learning	40
3.6	DT induction	42
3.6.1	The standard n-simplex	42
3.6.2	The splitting Rule	43
3.7	Gini index of diversity	46
3.8	Information Gain impurity measure	48
3.9	Misclassification rate impurity measure	50
3.10	Gini versus Gain	51

3.11	Variable combinations	51
3.12	Stopping Rule	52
3.13	Terminal nodes class assignation	54
3.14	Accuracy	54
3.15	Pruning	57
3.16	DT inference	58
3.17	Distance between decision trees	62
3.18	Instability coefficient	67
4	SODT (Soft Operators Decision Tree)	70
4.1	Introduction	70
4.2	Outline of the chapter and background	71
4.3	Definition SODT	73
4.4	Membership degree of an element $o \in \mathcal{X}$ to each of the SODT nodes	77
4.4.1	Measures of fuzzy cardinality	81
4.4.2	Properties of the degree of membership function μ	84
4.5	Guidelines for the learning and inference process of a SODT	93
4.6	Construction of the SODT	93
4.7	SODT induction	93
4.7.1	Function $\rho_m(j n)$ of probability	95
4.7.2	Proportion $\rho_{m,n}$ of objects going to node n	98
4.7.3	SODT node impurity function	99
4.7.4	Function ν of goodness	100
4.7.5	Relationship between crisp DT and SODT	102
4.7.6	Variable combinations: Oblique SODT	105
4.8	SODT Stopping Rule	105
4.9	SODT terminal node class assignment	107
4.10	Algorithm for the learning process of a SODT	108
4.11	SODT inference or classification	109
4.12	SODT Misclassification Rate	110
4.13	SODT for a fuzzified data sample	112
4.14	Distance between SODTs	114
5	Crisp DT vs. SODT	120
5.1	Split selection	120
5.2	Learning data	120
5.3	Split selection: goodness function	122
5.4	Noise	123
5.5	Preparation of sets for the study of variance	124
5.6	Study of variance	125
5.6.1	F-test of equality of two standard deviations	125
5.6.2	Example. Variance of the split points: SODT vs. crisp DT	125
5.7	Example: Simulated data \mathcal{D}_2	126

5.8	Example: Credit rating data	128
5.9	Conclusion	130
5.10	Distance between DTs and between SODTs	130
5.10.1	Simulated Data	131
5.10.2	Conclusion	133
5.11	Accuracy	133
5.11.1	Simulated data	135
5.11.2	Medical diagnosis data	135
5.11.3	Example: \mathcal{D}_3	137
5.12	Differentiability of function ν	139
6	Summary and conclusions	142
A	Fuzzy sets	145
A.1	Operation on fuzzy sets	147
B	Zermelo-Fraenkel set theory	149
C	Definitions	151

Notation

Symbols

$\mathcal{N}(0, 1)$:	Standard Normal Distribution
$\mathcal{N}(\mu, \sigma)$:	Normal Distribution with mean value μ and variance σ
\mathcal{F}_A or $\mathcal{P}(A)$:	Power set of A , which is composed of all its subsets
\mathcal{B} :	Borel σ -algebra on \mathbb{R}
\mathcal{B}^k :	Borel σ -algebra on the k -dimensional Euclidean space \mathbb{R}^k
$\mathcal{B}_{\mathcal{X}}$:	Borel σ -algebra on \mathcal{X}
std :	Standard deviation
Φ :	Distribution function of a $\mathcal{N}(0, 1)$
$F_{1-\alpha, \nu_1, \nu_2}$:	Critical value of the F distribution with ν_1 and ν_2 degrees of freedom and a significance level α
$P(A)$:	Probability of the set A
$P(A B)$:	Conditional probability of the set A given the set B
\mathbb{N} :	$\{0, 1, 2, \dots\}$
\mathbb{R} :	Set of real numbers
\mathbb{R}^d :	D-dimensional Euclidean Space
\mathbb{Z} :	$\{\dots, -2, -1, 0, 1, 2, \dots\}$
$\{0, 1\}^N$:	Vector with N boolean components
$I(\cdot)$:	Indicator function, which is equal to one if its argument holds and otherwise zero.
$\#$:	Denotes the cardinality of a set

$\ \cdot\ $:	Euclidean norm
$ \cdot $:	Absolute value
Δx :	Increment of x
$\{\mathbf{e}_1, \dots, \mathbf{e}_N\}$:	Standard/canonical basis of \mathbb{R}^N where $\mathbf{e}_1 = (1, 0, 0, \dots, 0),$ $\mathbf{e}_2 = (0, 1, 0, \dots, 0),$ \vdots $\mathbf{e}_N = (0, 0, 0, \dots, 1).$

Notation Decision Trees

χ :	Measurement space
MR_L :	Misclassification rate of the learning sample
MR_T :	Misclassification rate of the test sample

Notation Graph Theory

$ G $:	Order of a Graph $G = (V, E)$ ($\#V(G)$)
$\ G\ $:	Number of edges of a graph
$x \sim y; x, y \in V(G)$:	x vertex x is adjacent to the vertex y
$E(X, Y)$:	$X, Y \in \mathcal{P}(V(G))$: set of all $X - Y$ edges
$E(v)$:	Set of all edges in E at vertex v
$G_1 \simeq G_2$:	G_1 and G_2 are isomorphic
$G[V]$:	Induced subgraph of G (see definition 2.2.5)
$G - F$:	Graph G minus graph F (see definition 2.2.6)
$G + F$:	Graph G plus graph F (see definition 2.2.6)
$N_G(v)$:	Set of neighbours of v in G (see section 2.3)

Chapter 1

Introduction

The nowadays increasing number of fields where large quantities of data are collected generates an emergent demand for methods for extracting relevant information from huge databases. The “nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data” is called KDD (Knowledge Discovery in Databases). Within the KDD process, the step “that consists of applying data analysis and discovery algorithms that produce a particular enumeration of patterns (or models) over the data” is called data mining ([24], [48], [32], [29]).

For instance artificial neural networks and support vector machines are two examples amongst the various existing data mining models ([31], [55], [46], [13]). These models are very successful in numerous applications but, due to the complexity of the final model, they lack interpretability ([40]). In many applications, such as medical diagnosis, considering algorithms that learn accurately from data is not sufficient and making the discovery understandable by humans is equally important: the user can thus check whether the knowledge is correctly represented and modifications with additional expert information become then possible. A well known example of interpretable data mining models are decision trees ([10], [60], [52], [70]), which are widely used for classification purposes. Classification ([27]) refers to the data mining problem of attempting to predict the class of categorical data by building a model based on some predictor variables.

The interpretability of the final model of tree classifiers is due to the fact that they represent rules underlying data with hierarchical, sequential structures that recursively partition the data. The intuitive representation of classification rules is simple and easy to understand. Moreover, the factors in the rules are prioritized; that is, the most significant factors appear at the top levels of the tree.

A decision tree is induced on a *learning set*, which consists of *objects*. Each object is completely described by a set of *attributes/variables* and a *class label*. The set where these variables can take values in is called measurement space. The measurement space is partitioned in a data-driven manner based on the training set and the partition is presented in the form of a tree $T = (V, E)$, where the elements of V are called vertices or nodes of the tree T and the elements of E are its edges. At each internal node, a test is done to the set of attributes of an object to determine the child node the object belongs to. This test

is normally based on some conditions imposed to one attribute. The process is recursive because it can be repeated by treating each child node as a parent.

If we take the example of a numerical measurement space, at each internal node n of the tree, an *attribute* and a *threshold value* are responsible for the partitioning of this node. The elements belonging to n and fulfilling the condition “ $attribute \leq threshold\ value$ ” will belong to the left child node of n and the elements of n fulfilling “ $attribute > threshold\ value$ ” will belong to the right child node of n . The pair formed by the *attribute* and the *threshold value* is called *split point* at node n .

One of the main problems with tree classifiers is that they are very instable. As exposed in [38], the instability problem with tree-structured classifiers is that they may be sensitive to small changes in the learning sample. This instability issue complicates the process of the knowledge discovery because the users are disturbed by the different decision trees generated from almost the same input learning samples.

Up to now, research has mainly emphasized on the instability of the class prediction ([37], [8], [9])¹ and few studies focused on the instability of the tree with respect to its structure, which is important to provide insight into the data. A study on this field can be found in [38], where a stable classifier based on the concept of “almost equally good splits” is proposed. At each node, more than one attribute is used for its partition.²

Sometimes, it is important to consider one attribute at each node, since it gives an order of the priority of the attributes, the ones chosen at the first nodes are the ones which better divide the data into more homogeneous sets with respect to class assignation³. One important reason for the instability of a tree classifier is that the sharp partition of the space does not distinguish between elements which are much greater than the threshold value and elements close to it. This fact can lead to instability of the tree, in the sense that small changes in the training sample can cause big changes in the election of the split points at each node. For instance, a small change in the elements of the boundaries of the partition would lead to a wrongly classified sample and therefore, to a change on the split points chosen for the partition of the current node.

In order to avoid the instability produced by the sharp division of the measurement space, we propose “Soft Operators Decision Trees” (SODT). SODT is a new model that integrates decision trees with “soft operators”, mainly with the SFC operators proposed by Mlynski in his work [43]. The aim of this combination is to generate classifiers from training data through a process of recursively “soft” splitting of the data space. “Soft

¹In [8] and [9], for example, aggregation methods are incorporated in the tree classification algorithms to reduce the instability problem of the class prediction. These methods generate a set of classifiers and combine them to make a prediction. These methods obtain good accuracy, but rules are difficult to comprehend

²Ruey-Hsia Li ([38]) focuses the instability of tree classifiers on the fact that, when choosing a split, there are several which are almost equally good as the chosen one, and those will be the ones chosen when the data are slightly changed. To solve this problem, Ruey-Hsia Li proposes a model where at each node, a combination of the elements of the set of almost equally good splits are considered and a predicate is selected for this combination.

³An impurity function will measure the homogeneity of the data in a set, so, if all the data in a set belong to the same class, then this set has zero impurity.

splitting” in our case refers to a partition of the space where the operators \leq, \geq are not necessarily Boolean, but somehow fuzzified. Thus, the elements around the boundaries of the partition sets belong to these different sets with certain degrees of membership instead of purely belonging to one of them. This concept of *degree of membership* of an element to a *fuzzy set* (see appendix A) was introduced by Lotfi Zadeh in 1965 ([69]).

In our SODT, every element of the data space belongs to a node n with a certain *degree of membership*. This fuzzification is not done by using membership functions for the measurement space, i.e. it is not done by dividing the space into different fuzzy sets, but rather by applying membership functions for the operators responsible for the partition of each node.

Further advantages that SODT offers are:

- Ability to deal with noise: since SODT does not consider the data with exactly the same class they are given, but with degrees of membership to the different classes, therefore it does not adjust to the training data as much as a crisp classifier. We observe numerically that the tendency to overfitting is slightly smaller than in the crisp case.
- Ability to deal with uncertainty: at the end of chapter 4, we also explain a variant of SODT which deals with data given in a linguistic form with uncertainties. In this case, a two-class classification problem is considered and the learning sample is given as a set of features together with the degree of membership to one of the two classes. This type of data is called fuzzified data and occurs when it is impossible to assign a precise number to the observations in a sample.
- Possibility to apply gradient descent optimization methods: another aspect of the inclusion of soft operators is that the resulting goodness function is differentiable with respect to the parameters which have to be optimized, thus, allowing gradient descent methods to be used for this optimization.

1.1 Outline of this work

In this work, we are mainly concerned with the presentation of a new model that integrates decision trees (DT) with “soft/fuzzy comparison operators”. The aim of this combination is to generate a supervised learning technique which is a trade-off between the interpretability and accuracy of DTs and the structural stability and ability to deal with uncertainty that the fuzzy operators confer.

This work is organized as follows:

In chapter 2, we give a description of the topology of a SODT from the graph theory point of view. Since SODT is one of the “divide and conquer methods”, the underlying topology is a directed rooted binary tree and therefore we go in detail into its definition and propose an enumeration of its nodes. In this enumeration, a parent node n_i has a left child node n_{2i} and a right child node n_{2i+1} . This will set the foundation about notation

and enumeration we will work on in the next chapters since it draws a distinction between left and right child nodes as $\{n_i | i = 0 \pmod 2\}$ and $\{n_i | i = 1 \pmod 2\}$, respectively.

The bases of the new classification model presented in this work are decision tree classifiers. In chapter 3 we propose a new definition of them and of the manner they partition the measurement space \mathcal{X} . In this chapter we offer a new ordering at the categorical variables so that the same learning methods and definitions considered for numerical variables can be applied. We also draw a general scheme of the learning and inference process to create and apply decision trees for classification and explain both in details by combining methods extracted from the literature, adapted to our definition of DT.

The learning process consists of three main parts: the DT induction, the stopping rule and the terminal node class assignation. The DT induction refers to the construction of the tree, a rule for splitting at each node is necessary and for this reason *impurity functions* are defined. This functions, applied on the simplex Δ^{J-1} (for a J-class classification problem), are converted into *impurity measures* and the splitting point chosen to partition a node will be the one that maximizes the decrease of impurity. In sections 3.7, 3.8 and 3.9 we analyze the three main types of impurity measures. After the process of growing a tree, a stopping rule is needed to decide when to finish splitting and we present the main types. To finish the learning process, a class label is assigned to each terminal/leaf node. Eventually, a pruning process can be done in order to avoid overfitting and the accuracy of the decision tree is calculated by the estimation of the misclassification rate.

Further on, in section 3.16 we contribute to the DT inference process, which consists of the prediction of a class label to unclassified new instances, by offering an explicit formulation of the classifier/classification function represented by a decision tree.

At the end of this chapter, in sections 3.17 and 3.18, we propose an instability coefficient which is based on the notion of Lipschitz continuity. To compute this instability coefficient, we offer a metric to measure the proximity between decision trees and recommend several distances between databases from the literature.

This thesis converges towards its main part in chapter 4, where we are concerned with the presentation of our classification model (SODT). Mainly, we describe its construction, application and the consistency of the mathematical formulation behind this. An overview of the whole learning and inference process can be seen in figure 1.1, which is explained in detail in this chapter.

Among all the components of the learning process, the induction part is of special interest since a probability space is defined for SODT, which allows the use of the impurity functions described in the previous chapter to grow the tree. As we have already explained, in section 4.13 we present a variant of SODT which deals with data given with uncertainty, i.e., fuzzified data.

At the end of this chapter, concretely in section 4.14, we propose a distance to measure dissimilarities between the structure of two SODTs.

Finally, in chapter 5, we show the results of the implementation of SODT and compare numerically the stability and accuracy of a SODT and a crisp DT. For this implementation we have used simulated data, as well as data coming from medical diagnosis for heart disease and credit rating data.

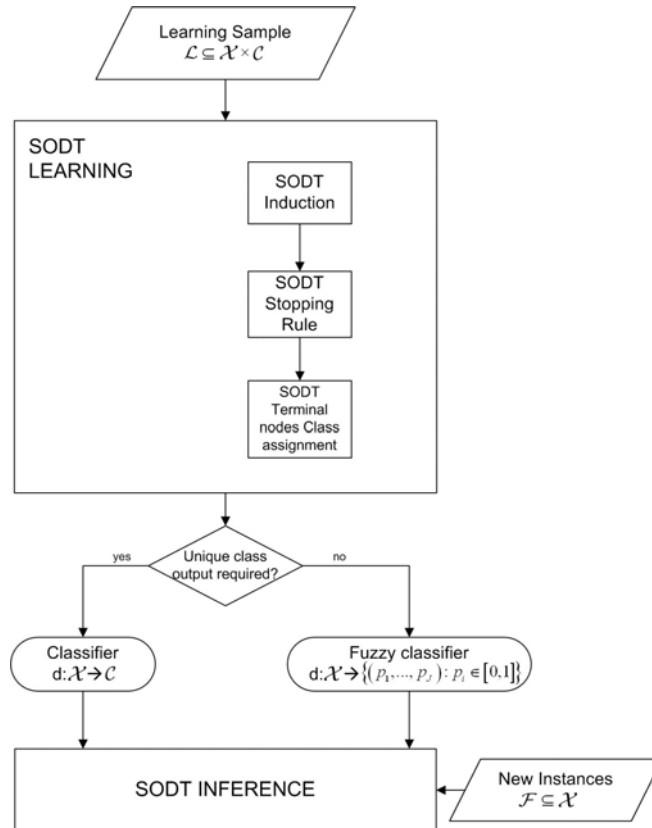


Figure 1.1: Outline of the learning and inference processes of a SODT

Chapter 2

Topology of the SODT

2.1 Introduction

The intention of this chapter is to provide a description of the topology of our model, trying to see a SODT (Soft Operators Decision Tree) from the graph theory point of view.

The basic idea involved in SODT is to break up a complex decision into a union of several simpler decisions, being thus one of the well known *divide and conquer* methods¹. The underlying topology of most of these methods, and of ours in concrete, is a *directed tree*. The simplest definition of a tree we find in graph theory is “A tree is an acyclic connected graph”. To describe and understand the topology of a SODT, we need to go in details into the definition of a *directed tree* as well as into the theorems about the enumeration of its nodes and levels, which will be very helpful to describe our method as well as for all the induction proofs.

Thus, this chapter will set the basis we will work on and it is divided in the following sections:

- basics of graph theory
- paths and cycles
- connectivity
- digraphs
- trees and forests

Most of the definitions and theorems are coming from graph theory and based on books like [51], [4], [14], [23], ... and others are proposed by us in order to complete all the scenario we need to have a reasonable topology of a SODT.

¹The technique is named “divide and conquer” because a problem is conquered by dividing it into several smaller problems.

Graph theory studies the properties of *graphs*, which are conceptually spatial configurations consisting of a finite set of points and a finite set of lines joining one point to another or even to itself and which may have an orientation (assigned direction). Graphs are natural mathematical models of physical situations in which the points represent either objects or locations and the lines represent connections. Applications of graphs are wide-ranging, in areas such as communications networks, ecology, engineering, operations research, set theory, information theory, sociology, etc.

Trees are a special type of graphs and also have applications in a wide variety of disciplines, particularly computer science. For example, they are widely applied to store data, to construct searching algorithms for finding a particular item in a list, as well as in decision theory.

In the following example we show an application of trees as graphical representation of a decision tree, which will be explained in detail in the next chapters (see [33]).

Example

In this example, the objective of classification is to decide whether a patient is at risk of having a heart disease. This decision is based on the measurements of the diastolic and systolic blood pressures, which are the features/variables.

The given learning sample is represented in the following table, where each row represents an *object* consisting of the measurements/attributes and the class of a patient. Patients who had a heart disease are considered as class 1, and patients who are healthy as class 0. The objective is to create a classifier which can predict the class of new patients by knowing their *systolic* and *diastolic* measurements.

<i>attribute</i> ₁ : Systolic	<i>attribute</i> ₂ : Diastolic	<i>Class</i>
172	71	1
168	90	1
194	51	1
150	92	0
192	60	0
152	80	1
163	79	1
180	71	0
185	70	0

A decision tree created from this learning sample is represented in figure 2.1 and the set of classification rules extracted from the tree is:

“IF $diastolic \leq 70 \wedge systolic \leq 192$ THEN *healthy*”

“IF $diastolic \leq 70 \wedge systolic > 192$ THEN *heart disease*”

“IF $diastolic > 70 \wedge systolic \leq 150$ THEN *healthy*”

“IF $diastolic > 70 \wedge systolic > 150$ THEN *heart disease*”.

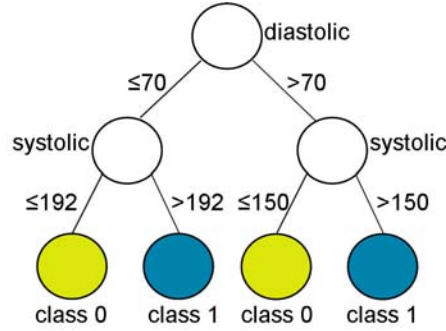


Figure 2.1: Example decision tree

2.2 Basics of graph theory

The birth of graph theory dates back to Leonard Euler (1707-1783), who studied the problem whether one could stroll around Königsberg and thereby cross each bridge across the Pregel exactly once. Abstracting this problem into a mathematical setting leads to the notion of a *graph*. While in the 18th and 19th century graph theory played only a marginal role within mathematics, in the 20th century the importance of graph theory dramatically increased, mainly due to the interplay with computer science.

Definition 2.2.1: A **graph** is an ordered pair $G = (V, E)$ of disjoint sets satisfying $E \subseteq [V]^2$; thus, the elements of E are 2-elements subsets of V . The elements of V are called **vertices** or nodes of the graph G , and the elements of E are its **edges**. A graph is usually represented by drawing a dot for each node and joining by a line each pair of dots whose corresponding nodes form an edge. It is irrelevant how these dots and lines are drawn, important is just the information which pairs of vertices form an edge and which do not.

A **graph on V** is every graph with vertex set V . Independently of the names we give to the previous V and E , the vertex set of a graph G is referred to as $V(G)$, and its edge set as $E(G)$. If x is a vertex of G , we sometimes write $x \in G$ instead of $x \in V(G)$.

The **order** of a graph G , written as $|G|$, is defined as its number of vertices $\#V(G)$, and $||G||$ denotes its number of edges. G is a **finite graph** if $|G| < +\infty$ and if not, then, G is called **infinite graph**; unless explicitly stated otherwise, we will just consider finite graphs because these are the ones we will need to study the topology of a DT and there

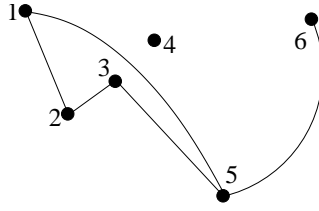


Figure 2.2: Example Graph $G = (V, E)$ where $V = \{1, \dots, 6\}$ and $E = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{3, 5\}, \{4, 5\}, \{5, 6\}, \{6, 1\}\}$

are many theorems coming from the graph theory which are only valid in the case of finite graphs.

For the empty graph (\emptyset, \emptyset) we simply write \emptyset . A graph is called **trivial graph** if its order is 0 (empty graph) or 1 (one vertex graph).

An edge $\{x, y\}$ (with $x \neq y$) is said to **join** the vertices x and y and is usually written as xy . Thus, xy and yx are said to be **equivalent**; the vertices x and y are the **endvertices** of this edge. If $xy \in E(G)$, then x and y are **adjacent**, or **neighbours**, vertices of G , and the vertices x and y are **incident** with the edge xy , in this case, it is said that $e = \{x, y\}$ is an edge at x (and at y). Two edges $e \neq f$ are **adjacent** if they have exactly one common endvertex. Also, $x \sim y$ means that the vertex x is adjacent to the vertex y . If all the vertices of G are pairwise adjacent, then G is **complete**. A complete graph on n vertices is a K^n ; a K^3 is called a **triangle**.

If $x \in X$ and $y \in Y$, then xy is an $X - Y$ edge and $E(X, Y)$ denotes the set of all $X - Y$ edges. To simplify the notation, instead of writing $E(\{x\}, Y)$ and $E(X, \{y\})$ we will write respectively $E(x, Y)$ and $E(X, y)$. $E(v)$ denotes the set of all the edges in E at vertex v .

Independent edges or vertices are the ones which are pairwise non-adjacent, then a set of vertices or of edges is called independent if no two of its elements are adjacent.

Definition 2.2.2: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. G_1 and G_2 are **isomorphic** if there exists a bijection $\varphi : V_1 \rightarrow V_2$ such that for all $x, y \in V_1$ it holds that

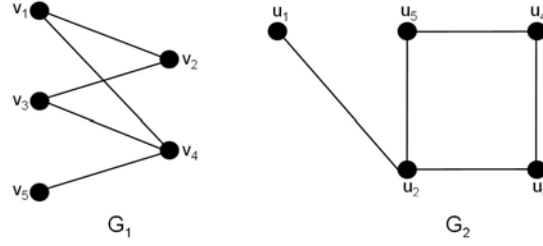
$$xy \in E_1 \Leftrightarrow \varphi(x)\varphi(y) \in E_2.$$

In this case, we write $G_1 \simeq G_2$ and the map φ is called an **isomorphism**. In the special case that $G_1 = G_2$, it is called **automorphism**.

Normally we do not distinguish between isomorphic graphs. Thus, we usually write $G_1 = G_2$ rather than $G_1 \simeq G_2$.

Example Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs as shown in figure 2.3. A bijection $\varphi : V(G_1) \rightarrow V(G_2)$ such that $\varphi(v_1) = u_5$, $\varphi(v_2) = u_4$, $\varphi(v_3) = u_3$, $\varphi(v_4) = u_2$ and $\varphi(v_5) = u_1$ is an isomorphism between the two graphs.

Definition 2.2.3: A map taking graphs as arguments is called a **graph invariant** if it assigns equal values to isomorphic graphs. The number of vertices and the number of

Figure 2.3: Two isomorphic graphs G_1 and G_2 .

edges of a graph are two simple graph invariants; the number of pairwise adjacent vertices is another.

Definition 2.2.4: For all pairs of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the union and intersection of graphs is defined as follows:

$$G_1 \cup G_2 := (V_1 \cup V_2, E_1 \cup E_2)$$

and

$$G_1 \cap G_2 := (V_1 \cap V_2, E_1 \cap E_2).$$

G_1 and G_2 are **disjoint** if $G_1 \cap G_2 = \emptyset$.

G_1 is a **subgraph** of G_2 (and G_2 a **supergraph** of G_1) if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$, and it is written as $G_1 \subseteq G_2$. We will also refer to that definition as G_2 **contains** G_1 .

Definition 2.2.5: Let $G_1 = (V_1, E_1)$ be a subgraph of a graph $G_2 = (V_2, E_2)$ ($G_1 \subseteq G_2$). G_1 is an **induced subgraph** of G_2 if E_1 contains all edges $xy \in E_2$ such that $x, y \in V_1$. It is said that V_1 induces or spans G_1 in G_2 and can be written as $G_1 =: G_2[V_1]$.

If $U \subseteq V_2$ is any set of vertices, then $G_2[U] = (U, E)$ is the graph on U where E is the set of elements of E_2 with both endvertices in U . A simplified notation is that if G is a subgraph of G_2 , then $G_2[V(G)]$ can be written $G_2[G]$.

Definition 2.2.6: Let $G = (V, E)$ be a graph, if U is any set of vertices, $G[V \setminus U]$ is normally written as $G - U$, i.e., $G - U$ is obtained by deleting the elements of $U \cap V$ and their incident edges. To simplify notations:

- if $U = \{u\}$, $G - \{u\}$ can be written as $G - u$,
- if H is a graph, $G - V(H)$ can be written as $G - H$.

Let F be a subset of $[V]^2$, then $G - F := (V, E \setminus F)$ and $G + F := (V, E \cup F)$. If $e \in [V]^2$, $G - \{e\}$ can be abbreviated by $G - e$ and $G + \{e\}$ by $G + e$.

2.3 The degree of a vertex

Note: Let $G = (V, E)$ be a non-empty graph. For all vertices $v \in V$, $N_G(v)$ denotes the set of neighbours of v in G . To simplify notations, we can write $N(v)$ instead of $N_G(v)$ if the graph G we are referring to is clear. More generally, the set of neighbours of a subset $U \subseteq V$ is composed of the neighbours in $V \setminus U$ of vertices in U and is denoted by $N(U)$.

Definition 2.3.1: Let G be a non-empty graph. For each vertex $v \in V(G)$, the **degree** $d_G(v) = d(v)$ of v is defined as the number of edges at v , $d(v) = \#E(v)$. In other words, it can be also defined as the number of neighbours of v .

An **isolated** vertex $v \in V(G)$ is a vertex of degree 0.

The concept “degree” is not just defined for vertices, but we can also obtain the **minimum**, **maximum** and **average degree** of a graph G as the numbers

$$\delta(G) := \min\{d(v) | v \in V\},$$

$$\Delta(G) := \max\{d(v) | v \in V\}$$

and

$$d(G) := \frac{1}{|V|} \sum_{v \in V} d(v)$$

respectively. It obviously holds that

$$\delta(G) \leq d(G) \leq \Delta(G).$$

If $\delta(G) = \Delta(G) = k$, then G is k -regular or **regular**, i.e., if the degrees of all vertices of G are identical.

2.4 Paths and cycles

As we asserted in the introduction of this chapter, the topology of our model is a directed tree. Since a tree is defined as an acyclic connected graph, in this section we go in detail into the definition of “acyclic” and the necessary foregoings.

Definition 2.4.1: A **path** is a non-empty graph $P = (V, E)$ of the form

$$V = \{x_0, x_1, \dots, x_k\} \quad E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\},$$

where all the x_i are distinct. The vertices x_0 and x_k are linked by P and are called its **ends**; the vertices x_1, \dots, x_{k-1} are the **inner** vertices of P .

The **length** of a path is given by its number of edges and P^k denotes a path of length k .

P is a path **from** x_0 to x_k (or **between** x_0 and x_k) and can also be written as the natural sequence of its vertices, $P = x_0x_1 \dots x_k$.

For the subpaths of P the following notation is used:

$$\begin{aligned} Px_i &:= x_0 \dots x_i \\ x_i P &:= x_i \dots x_k \\ x_i P x_j &:= x_i \dots x_j \end{aligned}$$

Let A, B be two sets of vertices, we call $P = x_0 \dots x_k$ an $A - B$ **path** if $V(P) \cap A = \{x_0\}$ and $V(P) \cap B = \{x_k\}$. As before, we write $a - B$ path rather than $\{a\} - B$ path, etc. Two or more paths are **independent** if none of them contains an inner vertex of each other. Two $a - b$ paths, for instance, are independent if and only if a and b are their only common vertices.

Given a graph H , we call P an H -**path** if P is non-trivial and meets H exactly in its ends. In particular, the edge of any H -path of length 1 is never an edge of H .

Definition 2.4.2: Let $P = x_0 \dots x_{k-1}$ be a path with $k \geq 3$, a **cycle** is a graph $C := P + x_{k-1}x_0$. A cycle is often denoted by its (cyclic) sequence of vertices, then the above cycle C can be written as $x_0 \dots x_{k-1}x_0$. The number of edges (or vertices) of a cycle is called its **length** and C^k denotes a cycle of length k and is called k -**cycle**.

Definition 2.4.3: A graph $G = (V, E)$ is **acyclic** if for all $G' \subseteq G$, there exists no cycle C isomorphic to G' (see definition 2.2.2), i.e, if it contains no subgraph isomorphic to a cycle.

2.5 Connectivity

As well as the previous section, this section describes one of the components of the tree definition. In concrete, here we define “connected graphs” and present an interesting proposition for the enumeration of its nodes.

Definition 2.5.1: A non-empty graph $G = (V, E)$ is called **connected** if two of its vertices are linked by a path in G (see definition 2.4.1). If $U \subseteq V(G)$ and $G[U]$ ² is connected, we also call U itself connected (in G).

Proposition 2.5.1: Let $G = (V, E)$ be a connected graph. The vertices of G can always be enumerated, say as v_1, \dots, v_n , so that $G_i := G[v_1, \dots, v_i]$ (see definition 2.2.5) is connected for every i .

Proof. By induction on i

$i = 1$: trivial

$i \longrightarrow i + 1$: Let us assume that v_1, \dots, v_i have been chosen for some $i < |G|$

²see definition 2.2.5 of induced subgraph

and $G_i := G[v_1, \dots, v_i]$.

Pick a vertex $v \in G - G_i$ (see definition 2.2.6).

As G is connected, it contains a $v - v_i$ path P (see definition 2.4.1).

Choose as v_{i+1} the last vertex of P in $G - G_i$.

Then v_{i+1} has a neighbour in G_i (see definition 2.2.1).

By induction, the connectedness of every G_i follows. \square

Note: In figure 2.4 we show an example of a connected graph enumerated by following the guidelines of proposition 2.5.1.

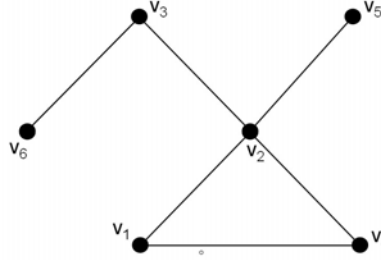


Figure 2.4: Example proposition 2.5.1

Definition 2.5.2: Let $G = (V, E)$ be a graph. A maximal connected subgraph of G is called a **component** of G . Note that a component, being connected, is always non-empty; the empty graph, therefore, has no components.

2.6 Digraphs

As we already mentioned in the introduction of this chapter, the tree representing the topology of every tree classifier, and of ours in concrete, is a directed tree. Therefore, in this section we describe directed graphs and its components.

Definition 2.6.1: A **directed graph** or **digraph** $G = (V, E)$ is composed of a set of vertices or nodes V and a set of edges $E \subseteq V \times V$. In other words, a digraph is a pair of disjoint sets (V, E) together with two maps

$$init : E \longrightarrow V \quad \text{and} \quad ter : E \longrightarrow V$$

assigning to every edge e an **initial vertex** $init(e)$ and a **terminal vertex** $ter(e)$.

The edge e is said to be **directed** from $init(e)$ to $ter(e)$.

Note: a directed graph may have several edges between the same two vertices x, y . Such edges are called **multiple edges** and they are called **parallel** if they have the same direction (say from x to y).

The edge e is called a **loop** if $init(e) = ter(e)$.

Definition 2.6.2: Let $G = (V, E)$ be a directed graph, the set of all **direct predecessors** of a node $i \in V$ is defined as $P(i) := \{j \in V : (j, i) \in E\}$, and the set of all its **direct successors** as $S(i) := \{j \in V : (i, j) \in E\}$. A vertex i with $P(i) = \emptyset$ is called a **source**, and a **sink or leaf** is a node i with $S(i) = \emptyset$.

In general, if G is a supergraph of a path P from i to j , then j is said to be a **successor** of i , and i is said to be a **predecessor** of j .

Definition 2.6.3: A directed graph D is an **orientation** of an undirected graph G if $V(D) = V(G)$ and $E(D) = E(G)$, and if $\{init(e), ter(e)\} = \{x, y\}$ for every edge $e = xy$. Intuitively, such an **oriented graph** arises from an undirected graph simply by directing every edge from one of its ends to the other. Thus, oriented graphs are directed graphs without loops or multiple edges.

2.7 Trees and forests

Definition 2.7.1: A graph $G = (V, E)$ is called **forest** if it is acyclic (see definition 2.4.3). Besides, if G is acyclic and connected (see definition 2.5.1), then it is called a **tree** and is normally denoted by $T = (V, E)$.

Assumption: Unless stated otherwise, all trees are assumed to be finite, i.e., to have a finite number of vertices.

Remark: We can say that a tree is a connected forest and a forest is a graph whose *components* (see definition 2.5.2) are trees.

Definition 2.7.2: Let $T = (V, E)$ be a tree, a vertex $v \in V$ is called an **end vertex** or **leaf** of T if the degree of v is equal to one (see definition 2.3.1).

Notation: In next chapters, we use the following notation to denote the set of the leaves of a tree:

If $T = (V, E)$ is a tree, then we set

$$V_{leaves}(T) = \{v \in V \mid d_T(v) = 1\},$$

where $d_T(v)$ denotes the degree of node v at tree T (see definition 2.3.1).

In next chapters we will refer to one single tree T . For simplification of notation, we usually omit T if the tree T we are referring to is clear. The notation is thus

$$V_{leaves} = \{v \in V \mid d(v) = 1\}.$$

Definition 2.7.3: The **eccentricity** of a vertex is the length of the longest simple path beginning at that vertex.

A **center** of a tree T is a vertex v with minimum eccentricity.

Definition 2.7.4: A **branch** T_t of a directed tree $T = (V, E)$ consists of node $t \in V$ and all the successors of t in T .

Lemma 2.7.1: Every non-trivial tree has at least two leaves.

Proof. It is trivial taking, for example, the ends of a longest path. \square

Remark: This fact is often used, especially in induction proofs about trees: if we remove an end vertex from a tree, the new graph is still a tree.

Lemma 2.7.2: Let $G = (V, E)$ be a forest on n vertices with $c \geq 1$ components. Then $\|G\| := \#E = n - c$.

Proof. We proceed by induction on $m = \|G\| := \#E$ (see definition 2.2.1). If $m = 0$ there is nothing to show. So, assume that $m \geq 1$. Choose an arbitrary edge $e \in E$ and consider what happens if we remove e from G . Clearly, the number of edges decreases by one. On the other hand, the number of components increases by one, as the subtree of G which contains e is split into two smaller trees. Then, the new $m' = m - 1$ and the new $c' = c + 1$.

By induction hypothesis we had $m = n - c$, which implies that $m - 1 = n - c - 1$. We can thus affirm that $m' = n - c'$. \square

Corollary 2.7.1: Every tree T on n vertices contains exactly $n - 1$ edges.

Proof. Trivial by application of previous lemma since the number of components of a tree is one. \square

Theorem 2.7.1: Let $G = (V, E)$ be a graph on n vertices. Then the following assertions are equivalent

- (i) G is a tree.
- (ii) For every pair $x, y \in V$ of distinct vertices G contains exactly one $x - y$ path.
- (iii) G is minimal connected (i.e, G is connected and for all $xy \in E$ the graph $G - xy$ is disconnected).
- (iv) G is maximal acyclic (i.e, G is acyclic and for all $x, y \notin E$ the graph $G + x, y$ contains a cycle).
- (v) G is acyclic and $|E| = n - 1$.
- (vi) G is connected and $|E| = n - 1$.

Proof. Assume G is a tree and let $x, y \in V$ be two distinct vertices. As G is connected, G contains at least one x, y path. This shows (i) \Rightarrow (ii). The implications (ii) \Rightarrow (iii) and (iii) \Rightarrow (iv) and (v) \Rightarrow (vi) follow immediatly from lemma 2.7.2. Finally, (vi) \Rightarrow (i) follows from the fact that $|E(T)| = n - 1$ by corollary 2.7.1. \square

Remark: The following corollary will help us later to deal with the notations we need for the formulation of our model as well as it will be essential for every proof made by induction.

Corollary 2.7.2: The vertices of a tree can always be enumerated, say as v_1, \dots, v_n , so that every v_i with $i \geq 2$ has a unique neighbour in $\{v_1, \dots, v_{i-1}\}$.

Proof. Use the numeration from Proposition 2.5.1 □

Corollary 2.7.3: If T is a tree and G is any graph with $\delta(G) \geq |T| - 1$, then $T \subseteq G$, i.e. G has a subgraph isomorphic to T (in definition 2.2.2 we announced that normally we do not distinguish between isomorphic graphs and therefore we usually write $G_1 = G_2$ rather than $G_1 \simeq G_2$).

Proof. Find a copy of T in G inductively along its vertex enumeration from corollary 2.7.2. □

2.7.1 Roots and orderings

Adding some extra structure to trees adapts them to applications in many disciplines, especially computer science. The bipartition of the space done by any tree classifier starts with the complete space, which will be represented as a vertex. In this case for example, it is convenient to consider one vertex of a tree as special, such a vertex is then called the **root** of this tree.

Definition 2.7.5: A **rooted tree** (T, r) is a tree T with a fixed distinguished vertex $r \in V(T)$, which is called the root of the tree.

Two rooted trees (T_1, r_1) and (T_2, r_2) are **isomorphic as rooted trees** if there is an isomorphism $\varphi : T_1 \rightarrow T_2$ such that $\varphi(r_1) = r_2$.

An example of a rooted tree can be seen in figure 2.5.

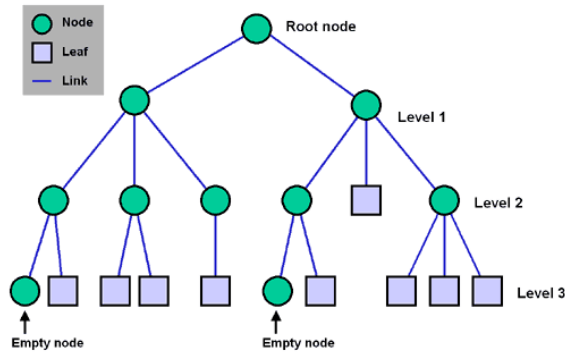


Figure 2.5: Example rooted tree

Definition 2.7.6: Let T be a tree and r a chosen root in T , a **tree-order** on $V(T)$ associated with T and r is the partial order on $V(T)$ imposed by letting

$$x \leq y \quad \text{if} \quad x \in rTy$$

Note: It is trivial to see that this relation satisfies the three properties of a partial order:

- Reflexivity: $x \leq x$ since xTx for all $x \in V(T)$
- Antisymmetry: If $x \leq y$ and $y \leq x$ for any $x, y \in V(T)$, then $x = y$
- Transitivity: If $x \leq y$ and $y \leq z$ for any $x, y, z \in V(T)$, then $x \leq z$

Note that r is the least element in this partial order, every leaf $x \neq r$ of T is a maximal element, the ends of any edge of T are comparable, and every set of the form $\{x | x \leq y\}$ (where y is any fixed vertex) is a **chain**, a set of pairwise comparable elements.

Note: From now on, we will consider rooted trees as directed rooted trees where the orientation will be given by the tree-order from definition 2.7.6. Thus, for a tree $T = (V, E)$, $xy \in E$ will imply that $x \leq y$.

2.7.2 Partition of a tree into levels

The main interest of this chapter lies in well defining all the environment of rooted directed trees in order to operate with them in next chapters. One of the most significant aspects is the partition of its nodes into levels and to accomplish this task we propose theorem 2.7.2.

Definition 2.7.7: Let (T, r) be a rooted tree, the **level** or **depth** of a vertex $v \in V(T)$ is defined as the length of the unique path from the root to this vertex v . In other words, the level of v is equal to the length of rPv (see section 2.4).

We define the **level or height of a rooted tree** (T, r) as the eccentricity of its root node r or, equivalently, as the maximum depth of any vertex.

The **distance** $\hat{d}_T(u, v)$ between two vertices $u, v \in V(T)$ is defined as the number of edges in the unique simple path between these vertices.

A tree with labels such as v_1, v_2, \dots, v_n assigned to its vertices is called a **labeled tree**.

Theorem 2.7.2: a) Let $G = (V, E)$ be a rooted (directed) tree. Then there exists a partition of $V = V_0 \dot{\cup} V_1 \dot{\cup} V_2 \dot{\cup} \dots \dot{\cup} V_L$, where L represents the *level of the tree* and each V_i contains the vertices/nodes at each level i of the tree, i.e., V_0 contains the root and V_L will contain leaves of the tree.

b) In other words, let $G = (V, E)$ be a rooted ditree, there exists a L-partition of V

$$V = V_0 \dot{\cup} V_1 \dot{\cup} V_2 \dot{\cup} \dots \dot{\cup} V_L$$

where L represents the *maximum level of the tree* and

$$\begin{aligned} & \forall e \in E, \quad \forall i \in \{0, \dots, L-1\} \\ & \text{if } \text{init}(e) \in V_i \implies \text{ter}(e) \in V_{i+1}. \end{aligned}$$

c) Equivalently,

$$\forall i \in \{0, \dots, L-1\} \quad \hat{d}_G(x, y) = 1 \quad \forall x \in V_i \quad \forall y \in V_{i+1}.$$

Proof. If $G = (V, E)$ is a directed rooted tree, then G is acyclic.

There exists an integer $k \geq -1$ and a natural partition of V into non-empty sets

$$V = V_0 \dot{\cup} V_1 \dot{\cup} V_2 \dot{\cup} \cdots \dot{\cup} V_{k+1}$$

which is constructed as follows:

Let us choose V_0 the set containing the source(s)/root

Remove from G all vertices in V_0 and all edges in $E_0 := \{e \in E \mid \text{init}(e) \in V_0\}$. The resulting graph $G_1 := G \setminus (V_0, E_0)$ is acyclic.

Again let V_1 be the set of all sources of the new graph G_1 and proceed as before.

Thus $i \in V_j$ if and only if the longest path from a source of G to i has length j . Moreover

$$i \in V_j \implies P(i) \subseteq V_0 \dot{\cup} \cdots \dot{\cup} V_{j-1} \quad \text{and} \quad S(i) \subseteq V_{j+1} \dot{\cup} \cdots \dot{\cup} V_{k+1}$$

□

We propose the following very simple lemma since we will require it in next chapters to refer to the set of all direct predecessors of a node as a unique node.

Lemma 2.7.3: Let $T=(V, E)$ be a rooted (directed) tree. Then for all $i \in V_j$, it is fulfilled that $\#\{k \mid k \in P(i)\} = 1$ and that $P(i) \subseteq V_{j-1}$, i.e., there exists a unique $k \in P(i)$ and $k \in V_{j-1}$.

Proof. $\#\{k \mid k \in P(i)\} = 1$ follows trivially from corollary 2.7.2 and $P(i) \subseteq V_{j-1}$ from theorem 2.7.2. □

Let us briefly define some concepts about rooted trees which are commonly used to describe decision trees.

Definition 2.7.8: A **child** of a vertex v in a rooted tree is a vertex that is the direct successor of v on a path from the root.

A **descendant** of a vertex v in a rooted tree is v itself or any vertex that is a successor of v on a path from the root.

A **proper descendant** of a vertex v in a rooted tree is any descendant except v itself.

The **parent** of a vertex v in a rooted tree is a vertex that is the direct predecessor of v on a path to v from the root. From lemma 2.7.3 we know that this direct predecessor exists and is unique.

The **parent function** of a rooted tree T maps the root of T to the empty set and maps every other vertex to its parent.

An **ancestor** of a vertex v in a rooted tree is v itself or any vertex that is the predecessor of v on a path to v from the root.

A **proper ancestor** of a vertex v in a rooted tree is any ancestor except v itself.

Siblings in a rooted tree are vertices with the same parent.

An **internal vertex** in a rooted tree is a vertex with children.

A leaf in a rooted tree can be thus defined as a vertex that has no children.

The **n th level** in a rooted tree is the set of all vertices at depth n .

2.7.3 Theorem for labeling binary rooted trees

A topological ordering of a digraph is a labeling of the vertices with consecutive integers so that every arc is directed from a smaller label to a larger label. In chapters 3 and 4 we will require, for each node, the labeling of its parent node. With this purpose, we propose theorem 2.7.3.

Since we set up theorem 2.7.3 for complete binary rooted trees and rooted trees were explained in section 2.7.1, let us define previously a complete binary tree.

Definition 2.7.9: An m -ary tree is a rooted tree such that every internal vertex has at most m children.

A **full m -ary tree** is a rooted tree such that every internal vertex has exactly m children.

A **complete m -ary tree** is an m -ary tree in which every parent has m children and all leaves are at the same depth.

Definition 2.7.10: A (pure) **binary tree** is a rooted tree such that every internal vertex has at most two children. This meaning of binary tree occurs commonly in pure graph theory.

A **binary tree** is a tree with the following properties:

- Each internal node has two children
- The children of a node are an ordered pair

We call the children of an internal node **left child** and **right child**. This meaning of binary tree occurs commonly in computer science and in permutation groups.

Alternative recursive definition: a binary tree is either:

- a tree consisting of a single node, or
- a tree whose root has an ordered pair of children, each of which is a binary tree.

A **complete binary tree** is a binary tree in which every parent has two children and all leaves are at the same depth.

In order to correctly define a decision tree classifier, we need a uniquely determined labeling of the parents of a node. This is the reason why we extend proposition 2.5.1 to the following theorem:

Theorem 2.7.3: Let $T = (V, E)$ be a complete binary rooted tree. Then, there exists a labeling of its nodes such that for all $j \in \{1, \dots, \frac{|T|-1}{2}\}$ (i.e. $v_j \in V \setminus V_{leaves}$) it is fulfilled that $(v_j, v_{2j}) \in E$ and $(v_j, v_{2j+1}) \in E$ ³, as well as that $T_i := G[v_1, \dots, v_i]$ is connected for every $i \in \{1, \dots, |T|\}$.

³i.e., $E = \{(v_i, v_j) \in (V \setminus V_{leaves}) \times V \mid j = 2i \vee j = 2i + 1\}$

Proof. By induction on i

$i = 1$: trivial

$i \longrightarrow i + 1$:

Pick vertex root as v_1 , and assume inductively that v_1, \dots, v_i have been chosen for some $i < \frac{|T|-1}{2}$ and $T_i := T[v_1, \dots, v_i]$. Now pick a vertex $v \in T - T_i$. As T is connected, it contains a $v - v_i$ path P . Choose as v_{i+1} the last vertex of P in $T - T_i$; then v_{i+1} has a neighbour in T_i . The connectedness of every T_i follows by induction on i .

Introducing the modification that every time we pick a vertex, the next nodes to be labeled are the ones with distance 1 to the chosen vertex, then the two conditions $(v_j, v_{2j}) \in E$ and $(v_j, v_{2j+1}) \in E$ are fulfilled since T is a binary tree. \square

Note: As an illustration of the labeling of a complete binary rooted tree proposed in this theorem see figure 2.6.

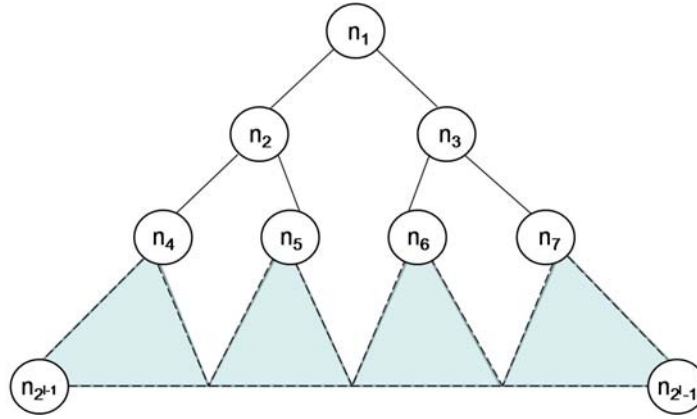


Figure 2.6: Labeling of a complete binary rooted tree of level L

2.7.4 Other notions of trees

Definition 2.7.11: An **ordered tree** is a rooted tree in which the children of each internal vertex are linearly ordered.

A **left sibling** of a vertex v in an ordered tree is a sibling that precedes v in the ordering of v and its siblings.

A **right sibling** of a vertex v in an ordered tree is a sibling that follows v in the ordering of v and its siblings.

A **plane tree** is a drawing of an ordered tree such that the left-to-right order of the children of each node in the drawing is consistent with the linear ordering of the corresponding vertices in the tree.

In the **level ordering** of the vertices of an ordered tree, u precedes v under any of these circumstances: if the depth of u is less than the depth of v ; if u is a left sibling of v ;

if the parent of u precedes the parent of v .

Two ordered trees (T_1, r_1) and (T_2, r_2) are **isomorphic as ordered trees** if there is a rooted tree isomorphism $f : T_1 \rightarrow T_2$ that preserves the ordering at every vertex.

Definition 2.7.12: The **principal subtree** at a vertex v of a rooted tree comprises all descendants of v and all edges incident to these descendants. It has v designated as its root.

The **left subtree** of a vertex v in a binary tree is the principal subtree at the left child. The right subtree of v is the principal subtree at the right child (see figure 2.7).

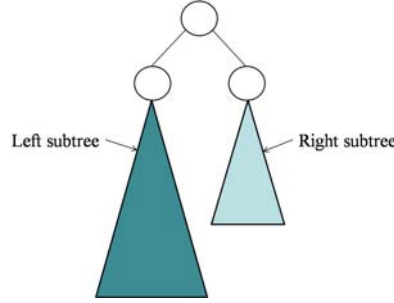


Figure 2.7: Subtrees rooted at the children of the root node

A **balanced tree of height h** is a rooted m -ary tree in which all leaves are of height h or $h-1$.

A **decision tree** can be roughly defined as a rooted tree in which every internal vertex represents a decision and each path from the root to a leaf represents a cumulative choice.

Theorem 2.7.4: A binary tree T of order $|T| = n$ has height at least $\lfloor \log_2 n \rfloor$.

Proof. By induction on the order of the tree (number of nodes of the tree, n).

Consider the base case, $n = 1$, then, the tree has 0 edges (R.A. If a tree T with $|T|=1$ has some edge, then we would have a loop in the tree). It can be proved applying corollary 2.7.1, then, a tree with 1 vertex contains exactly $||T|| = 1 - 1 = 0$ edges.⁴ This implies that the height of the tree is equal to zero, which fulfills the statement, since $\log_2 1 = 0$

Inductive step: Suppose that the statement is true for all $n \in \mathbb{N}$, $n \leq k$, i.e, for every binary tree T with $|T| = n \leq k$, it is verified that height of T is greater or equal than $\lfloor \log_2 n \rfloor$.

Consider any binary tree T' such that $|T'| = k + 1$. Consider the two sub-trees T'_L and T'_R rooted at the left and right children of the root node, as shown in figure 2.7. The fact that $|T'| = k + 1$ and one node is the root, implies that the total number of nodes in the

⁴Recall that $|G|$ =order of a graph, $||G||$ = number of edges

two subtrees is k . We assume, without loss of generality, that the left-subtree has at least $\frac{k}{2}$ nodes (or $\frac{k+1}{2}$), i.e, $|T'_L| \geq \frac{k+1}{2}$.

The height of tree T' is at least equal to the height of T'_L plus 1, corresponding to the root node.

Since $\frac{k+1}{2} \leq k$ for all $k \geq 1$, we can apply the inductive hypothesis to T'_L and, so, its height must be greater or equal than $\lfloor \log_2 \frac{k+1}{2} \rfloor$. Then, the height of T' is at least $\lfloor \log_2 \frac{k+1}{2} \rfloor + 1$, which is equal to $\lfloor \log_2(k+1) \rfloor$.

$$\begin{aligned} \lfloor \log_2 \frac{k+1}{2} \rfloor + 1 &= \lfloor \log_2(k+1) - \log_2(2) \rfloor + 1 \\ &= \lfloor \log_2(k+1) - 1 \rfloor + 1 \\ &= \lfloor \log_2(k+1) \rfloor \end{aligned}$$

□

Chapter 3

Decision trees

Since the basis for the classification model presented in this work are decision tree classifiers, we are concerned with the presentation of them in this chapter. Here, we redefine the main concepts in a more precise form in order to have a mathematical framework that can be used in future works and that we use in chapter 4 as basis for our model.

3.1 Introduction

Decision trees, also named classification trees, are one of the most popular methods for learning and reasoning from feature based examples. In several fields, such as medical diagnosis for example, it is not just to consider algorithms that learn from data, but also the way the knowledge is obtained is important. Thus the user does not only learn about the problem of interest, but can also check whether the knowledge represented within the learning system is correct or at least plausible. A good example are decision trees, which are based on discriminative learning algorithms working by means of recursive partitioning. The data space is partitioned in a data-driven manner and the partition is represented in form of a tree (see figure 3.1). A decision tree can be transformed into a rule base, by following each path from the root node to a terminal node. The induction algorithm that creates the decision tree has a preference for shorter trees. Thus the system is not just transparent but also compact and so easier to comprehend.

One of the main properties of decision trees is their capability to break down a complex decision-making process into several simpler decisions, providing in this way an easily interpretable solution.

Tree-based methods date back from work in the social science by Morgan & Sonquist (1964) and Morgan & Messenger (1973). Advances in practical and theoretical aspects were introduced by Breiman, Friedman, Olshen and Stone (1984) in their work [10]. Their classification and regression tree (CART [10]) algorithm brought tree-based methods to the attention of statisticians. Created for numerical domains, CART builds binary trees by means of splitting according to the Gini impurity measure criteria and finally allows a

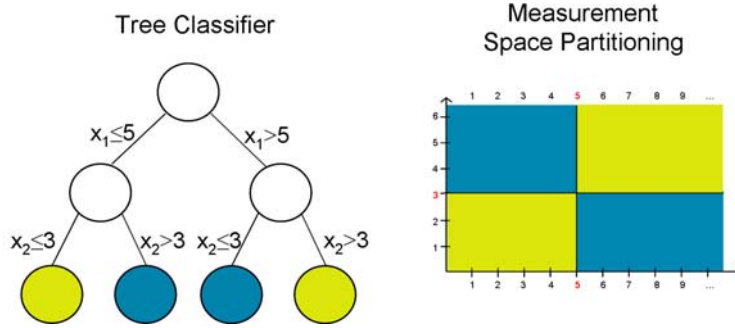


Figure 3.1: Example of the binary partitioning of the space made by a decision tree

pruning phase to adjust the size of the tree and make a compromise between accuracy and complexity. Another well-known tree based algorithm for classification problems is ID3 (Quinlan [52], 1986), which works for symbolic domains with small cardinality. At every node the less entropy attribute is selected to split, this process is repeated recursively and small trees are preferred. Figure 3.2 illustrates an example of a decision tree generated by ID3 and by CART.

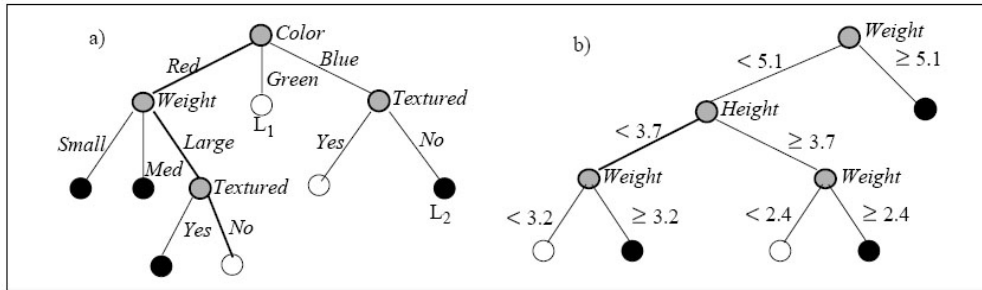


Figure 3.2: Example of a decision tree generated by ID3(a) and CART(b)

3.2 Outline of the chapter

The current chapter is structured as follows:

In section 3.3 some notations and definitions are presented, which are needed for the description of binary tree structured classifiers in section 3.4. In this section we also define the representation of the partition of the measurement space originated by splitting at each internal node of a decision tree. In its subsection 3.4.1 we establish an ordering of the categorical variables, which enables us to develop a formal definition of a DT that can be used for ordered as well as for categorical variables, consisting of the assignation of a variable and a threshold value to split at each internal node. We offer this formal definition in subsection 3.4.2.

In section 3.5 the process of learning a DT from a given data sample is presented and divided into three main parts: the DT induction (sections 3.6-3.11), the stopping rule (section 3.12) and the terminal node class assignation (section 3.13). We summarize this processes for the construction of a decision tree, together with the inference process for its application, in figure 3.9.

CART ([10]) proposes a "pruning process" after the construction of a DT and it will be described in section 3.15. In order to compute the accuracy of a decision tree, some methods for estimation of the misclassification rate are depicted in section 3.14.

In section 3.16 we contribute to the DT inference process, which consists of the prediction of a class label to unclassified new instances, by offering an explicit formulation of the classifier/classification function represented by a decision tree.

We notice that two classifiers which are equal w.r.t. class prediction do not necessarily come from the same decision tree structure and, for this reason, we propose a distance in section 3.17 to measure the differences between the structures of decision trees. This distance, together with the Lipschitz constant, will be the basis of the conceptional approach for dealing with stability questions that we propose in section 3.18.

3.3 Notation and definitions

Definition 3.3.1: A learning sample \mathcal{L} is a set of $(h + 1)$ -tuples

$$\mathcal{L} = \{(x^{(k)}, y^{(k)}) : k \in \{1, \dots, N\}\}. \quad (3.1)$$

where:

- each $x^{(k)}$ is a **measurement vector**, $x^{(k)} \in \mathcal{X}$, being $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_h$ the **measurement space**. The components of the vectors $x^{(k)}$ can be viewed as attributes and h is then the number of attributes. Attributes are also referred to as features, predictors or independent variables. Each measurement vector $x^{(k)}$ has the form

$$x^{(k)} = (x_1^{(k)}, \dots, x_h^{(k)}).$$

- If we have a classification problem, $\mathcal{C} := \{c_1, c_2, \dots, c_J\}$ is the set of classes and it can be represented as:

$$\mathcal{C} := \{1, \dots, J\}$$

in a J-class problem.

- For all $k \in \{1, \dots, N\}$, $y^{(k)} \in \mathcal{C}$ is the class label assigned to the measurement vector $x^{(k)}$. A class label is also known as dependent variable or category.
- Each input-output pair $(x^{(k)}, y^{(k)})$ is called an **object**. An object is completely described by a set of attributes and a class label.
- N is the number of objects in \mathcal{L} .

- A concept $g : \mathcal{X} \longrightarrow \mathcal{C}$ underlying a learning sample is a surjective function which is the true mapping between the measurement space and the set of classes. A noise free learning sample is one in which all the objects are generated using the underlying concept. Thus, a noisy free learning sample can be represented as:

$$\mathcal{L} = \{(x^{(k)}, g(x^{(k)})) : k \in \{1, \dots, N\}\}. \quad (3.2)$$

Remark: We describe thus a learning sample \mathcal{L} as a relation $\mathcal{L} = (X, C) \subseteq \mathcal{X} \times \mathcal{C}$ that assigns a class in \mathcal{C} to each measurement vector $x \in \mathcal{X}$ and can then be viewed as a matrix $\mathcal{L} \sim \mathcal{M}_{N \times (h+1)}$.

3.3.1 Types of variables

The word “attribute” denotes the parameters that describe the input information of an object of the learning sample \mathcal{L} . There are different types of attributes or variables which can be used to describe an element $x \in \mathcal{X}$.

We distinguish two general types of variables that can appear in the measurement vector:

- Ordered or numerical variables
- Categorical variables

Definition 3.3.2: A variable x_i , for $i \in \{1, \dots, h\}$, is called **ordered** or **numerical** if its measured values are real numbers, i.e., if $\mathcal{X}_i \subseteq \mathbb{R}$. These are further distinguished into **ordinal** variables and **continuous** variables. Ordinal variables are discrete variables that have a natural order for the values they can assume, i.e, x_i is called an ordinal variable if $\mathcal{X}_i \subseteq \mathbb{N}$. Examples are age, number of floors in buildings or ratings in questionnaires, etc, ...

Continuous variables can assume any value from an interval of the real numbers. It means that the values a continuous variable can take are the elements of \mathcal{X}_i , which can be of the form:

$$[a, b], \quad [a, b), \quad (a, b], \quad (a, b), \quad (a, +\infty), \quad [a, +\infty), \quad (-\infty, b), \quad (-\infty, b]$$

for $a, b \in \mathbb{R}$. Examples are distances, durations of time, etc, ...

Definition 3.3.3: A variable x_i , for $i \in \{1, \dots, h\}$ is **categorical** if it takes values in a finite set not having any natural ordering. Variables belonging to this class assume values belonging to different categories, which do not have an inherent order, and which cannot be used for calculations, even if encoded as numbers. A categorical variable, for instance, could take values in a set of colours.

A special type of categorical variables are the **binary** variables, which are categorical values but that can only take two possible values, they can also be represented as numerical values, e.g. 0 and 1. Examples are the gender of persons or indicators for membership of classes.

Finally, we consider the following definition

Definition 3.3.4: If all measurement vectors $x^{(k)}$ are of fixed dimensionality, we say the data have **standard structure**.

In definition 3.3.1 the measurement vectors are supposed to have standard structure, because we fixed with h the number of attributes to be measured.

3.3.2 Classifiers as partitions

A systematic way of predicting class membership is a rule that assigns a class membership in \mathcal{C} to every measurement vector x in \mathcal{X} . That is, given any $x \in \mathcal{X}$, the rule assigns one of the classes $\{1, \dots, J\}$ to x .

Definition 3.3.5: A **classifier** or **classification rule** is a function $d(\cdot)$ defined on \mathcal{X} so that for every x , $d(x)$ is equal to one of the elements of the set $\mathcal{C} = \{1, 2, \dots, J\}$. $d(x)$ represents the **class label** of the measurement vector $x \in \mathcal{X}$.

Another way of looking at a classifier is to define A_j as the subset of \mathcal{X} on which $d(x) = j$; that is,

$$A_j = \{x \mid d(x) = j\}$$

The sets A_1, \dots, A_J are disjoint and $\mathcal{X} = \bigcup_j A_j$. Thus, the A_j form a partition of \mathcal{X} . This gives the equivalent

Definition 3.3.6: A classifier is a partition of \mathcal{X} into J disjoint subsets A_1, \dots, A_J , $\mathcal{X} = \bigcup_j A_j$ such that for every $x \in A_j$ the predicted class is j .

Classification problem The aim is to construct a classifier which represents the concept underlying a learning sample \mathcal{L} , i.e., the true mapping between the measurement space \mathcal{X} and the set of classes \mathcal{C} .

The problem can be stated as follows: given a learning sample $\mathcal{L} = \{(x^{(k)}, y^{(k)}) : k \in \{1, \dots, N\}\}$, produce a classifier $h : \mathcal{X} \rightarrow \mathcal{C}$ which maps an instance $x \in \mathcal{X}$ to its class label $y \in \mathcal{C}$ ¹.

With this purpose, tree classifiers partition the measurement space in a data driven manner and assign a class label to each set in the partition. In the next sections we will explain these processes in detail.

3.4 Binary tree structured classifiers

Normally, *binary tree structured classifiers* (see 2.7.10) are just called *tree structured classifiers* and their construction consists in dividing the subsets of the measurement space \mathcal{X} by splitting them repeatedly into two descendant subsets. The root node (see definition 2.7.5) of the tree is the complete measurement space \mathcal{X} and the leaves (see definition 2.7.2) of the

¹Some authors ([55], [33]) define a classification problem as the pair $(\mathcal{X}, \mathcal{C})$.

tree form a partition of \mathcal{X} . Moreover, for any node of the tree, its children form a partition of this node.

Each terminal node is designated by a class label $j \in \mathcal{C} = \{1, \dots, J\}$. The partition A_1, \dots, A_J (see definition 3.3.5) corresponding to the classifier is obtained as the union of all terminal subsets corresponding to the same class.²

Remark: We just consider binary trees (binary splitting) instead of general trees (multiway splitting) because any general tree can be trivially converted into a binary one (as an example, see figure 3.3) and normally, the multiway splitting does not allow further splitting on a variable/attribute once it has been used. This fact can lead to unparsimonious trees with a higher number of leaves, as it is illustrated in figure 3.4 (see [66] and [50]).

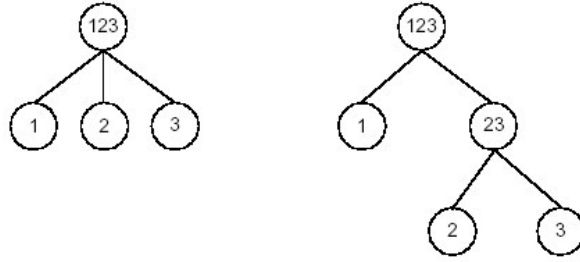


Figure 3.3: 3-ary tree converted to binary tree.

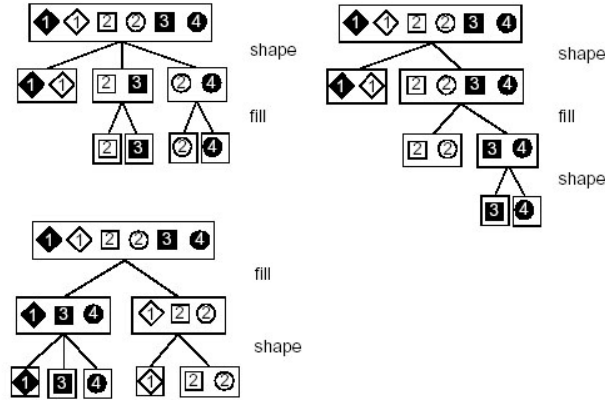


Figure 3.4: Multibranch trees (left) vs. binary tree (right).

Definition 3.4.1: A (binary) tree structured classifier is a classifier $d : \mathcal{X} \longrightarrow \mathcal{C}$ whose partition of the measurement space is represented as a rooted (binary) tree (T, r) (see section 2.7.1).

²From now on, we take the notation from graph theory and then a node n_i substitutes the subset \mathcal{X}^i for all $\mathcal{X}^i \subseteq \mathcal{X}$.

We propose the following definition of this representation in form of a rooted tree:

Definition 3.4.2: The representation of the partition of the measurement space \mathcal{X} created by a tree structured classifier is in form of a rooted tree (T, r) (see section 2.7.1) and has the following structure:

- $r = \mathcal{X}$
- $T = (V, E)$ where $V \subseteq \mathcal{P}(\mathcal{X})$ (for $\mathcal{P}(\mathcal{X})$ the power set of \mathcal{X}) and the fact that $(A, B) \in E$ implies that $B \subseteq A$.
- Let us consider the set $S(A)$ of the direct successors of A (see definition 2.6.2), then the elements of $S(A)$ form a partition of A , i.e.,

$$\bigcup_{x \in S(A)} x = A$$

and $x \cap y = \emptyset$ for all $x, y \in S(A)$. (If the tree structured classifier is binary, then $\#S(A) \in \{0, 2\}$ for all $A \in V$.)

- Let $V_{leaves} \subseteq V$ be the subset of the set of nodes of the tree $T = (V, E)$ with degree 1 (see definition 2.7.2) and let $l : V_{leaves} \rightarrow \mathcal{C}$ be the map that designates a class label $j \in \mathcal{C}$ to each terminal node, then

$$A_j = \bigcup \{\mathcal{X}^i \mid \mathcal{X}^i \in V_{leaves} \wedge l(\mathcal{X}^i) = j.\}$$

Example

The process of repeated splitting of subsets of \mathcal{X} into descendant subsets described above is pictured in figure 3.5 for a hypothetical three-class tree.

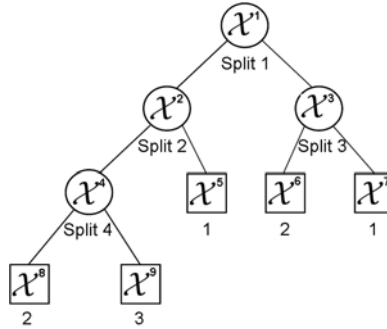


Figure 3.5: Representation of a partition of a measurement space \mathcal{X} in form of a tree.

In this figure, $\mathcal{X}^2 \cup \mathcal{X}^3 = \mathcal{X}$ and \mathcal{X}^2 and \mathcal{X}^3 are disjoint. Both observations hold similarly for all subsets of \mathcal{X} in the tree and its direct successors³.

³see definition 2.6.2

The terminal subsets ($\mathcal{X}^8, \mathcal{X}^9, \mathcal{X}^5, \mathcal{X}^6$ and \mathcal{X}^7) are indicated by a rectangular box and they form a partition of \mathcal{X} , i.e.

$$\mathcal{X} = \mathcal{X}^8 \cup \mathcal{X}^9 \cup \mathcal{X}^5 \cup \mathcal{X}^6 \cup \mathcal{X}^7$$

and $\mathcal{X}^i \cap \mathcal{X}^j = \emptyset$ for all $i, j \in \{8, 9, 5, 6, 7\}$ $i \neq j$.

Each terminal subset is designated by a class label, in this example, \mathcal{X}^8 is designated by the class label 2, \mathcal{X}^9 by 3, \mathcal{X}^5 by 1, \mathcal{X}^6 by 2 and \mathcal{X}^7 by 1. The partition corresponding to the classifier is:

$$A_1 = \bigcup \{\mathcal{X}^i \mid \mathcal{X}^i \in V_{leaves} \wedge label(\mathcal{X}^i) = 1\} = \mathcal{X}^5 \cup \mathcal{X}^7$$

$$A_2 = \bigcup \{\mathcal{X}^i \mid \mathcal{X}^i \in V_{leaves} \wedge label(\mathcal{X}^i) = 2\} = \mathcal{X}^6 \cup \mathcal{X}^8$$

$$A_3 = \bigcup \{\mathcal{X}^i \mid \mathcal{X}^i \in V_{leaves} \wedge label(\mathcal{X}^i) = 3\} = \mathcal{X}^9$$

If we change the terminology to that of tree theory, then a node n is a subset of \mathcal{X} , being n_1 the measurement space \mathcal{X} . Terminal and nonterminal sets become terminal and nonterminal nodes respectively.

3.4.1 How does split decomposition work?

To build a decision tree it is necessary to find a split on the data at each internal node (see [16]).

Definition 3.4.3: Given a finite set X , a **split** on X is a bipartition of X into two non-empty subsets A, B .

The main idea behind split decomposition is to construct, for a data set X , splits $X = A \dot{\cup} B$, which separate the samples of X into two more homogeneous subsets w.r.t. class assignation (see section 3.6).

Let us suppose that the data have standard structure. The measurement vectors have then the form $x = (x_1, \dots, x_h)$. Each variable x_i , $i = 1, \dots, h$, can be either of ordered or categorical type.

We are considering **univariate splits**, it means that each split depends on the value of one single variable, which is called **split variable**. A description of multivariate splits is given in section 3.11.

For each ordered or numerical variable x_i , the natural univariate splits on any data set $X \subseteq \mathcal{X}$ are of the form

$$A = \{x \in X \mid x_i \leq \sigma\}$$

$$B = A^c = \{x \in X \mid x_i > \sigma\}$$

for a $\sigma \in \mathbb{R}$. The variable x_i is then the split variable and the value σ is called **threshold value** of the split variable.

For a categorical variable x_i , let $D = \{D_1, \dots, D_M\}$ be the finite set where the variable takes values in. Then, the univariate splits on X would be of the form

$$A = \{x \in X \mid x_i \in S\}$$

$$B = \{x \in X \mid x_i \notin S\}$$

for each $S \in \mathcal{P}(D) \setminus \{\emptyset, D\}$.

Theorem 3.4.1: Let $X \subseteq \mathcal{X}$ be a finite subset of the measurement space, then the number of possible splits of X based on a numerical variable is finite.

Proof. x_i numerical. Then, the elements of \mathcal{L} contain at most $\#\mathcal{L}$ distinct values $x_i^{(1)}, \dots, x_i^{(\#\mathcal{L})}$ of x_i . Without loss of generality we can consider $x_i^{(1)}, \dots, x_i^{(\#\mathcal{L})}$ ordered so that $x_i^{(1)} \leq x_i^{(2)} \leq \dots \leq x_i^{(\#\mathcal{L})}$. There are at most $\#\mathcal{L}$ different splits of the form

$$A = \{x \in X \mid x_i \leq \sigma\}$$

$$B = A^c = \{x \in X \mid x_i > \sigma\}$$

and these are given by

$$A = \{x \in X \mid x_i \leq \sigma_j\}$$

$$B = A^c = \{x \in X \mid x_i > \sigma_j\}$$

for $j = 1, \dots, \#\mathcal{L}$ and where σ_j is taken, for example, as the mid-point of two consecutive values of x_i , $\sigma_j = \frac{x_i^{(j)} + x_i^{(j+1)}}{2}$.⁴ □

Theorem 3.4.2: Let $X \subseteq \mathcal{X}$ be a finite subset of the measurement space, then the number of possible splits of X based on a categorical variable is finite and equal to $2^{M-1} - 1$, where M is the number of categories.

Proof. x_i categorical. If x_i takes values on the set $D = \{D_1, \dots, D_M\}$ with M distinct values, then the dimension of the power set of D is $\#\mathcal{P}(D) = 2^M$. This can be easily proved using binomial coefficients:

$$\#\mathcal{P}(D) = \binom{M}{0} + \binom{M}{1} + \dots + \binom{M}{M} = (1+1)^M = 2^M$$

$\#\{\mathcal{P}(D) \setminus \{\emptyset, D\}\} = 2^M - 2$. But since the splits $\{A = S, B = S^c\}$ and $\{A = S^c, B = S\}$ are considered to be the same for all $S \in \mathcal{P}(D)$, then the number of possible splits is $\frac{2^M - 2}{2} = 2^{M-1} - 1$. □

In proposition 3.4.1 it can be observed that this number can be reduced.

⁴Every threshold value between two consecutive elements of a set X will give the same split on X .

Proposition 3.4.1: ⁵ Suppose $J = 2$ (i.e. a two-class problem) and $i(p)$ ⁶ strictly concave. For a categorical feature, order the levels in increasing $p(1|x = x_i)$ ⁷. Then a split of the form $A = \{x_1, \dots, x_l\}$, $B = \{x_{l+1}, \dots, x_L\}$ maximizes the goodness function ⁸.

Proposition 3.4.1 reduces the number of splits which need consideration for two classes from $2^{L-1} - 1$ to $L - 1$. The original proof of this theorem is given by Breiman *et al* and a very much shorter proof is given by Ripley in his work [55].

Definition 3.4.4: For a two-class problem, let us consider the two classes labelled by class 1 and class 2, and let x be a categorical variable of the measurement vector. Based on the previous proposition 3.4.1, we define a binary relation \preceq on the possible values of x . This relation is established in the following way:

Consider $a \preceq b$ if and only if $p(1|x = a) \leq p(1|x = b)$ for all $a, b \in \mathcal{D}(x)$, where $\mathcal{D}(x)$ is the domain of definition of variable x .

Proposition 3.4.2: Let us consider the same assumptions as in definition 3.4.4 and furthermore, let us suppose $p(1|x = a) \neq p(1|x = b)$ for all $a, b \in \mathcal{D}(x)$. Then the relation \preceq defines an order on the domain of definition of variable x , i.e. on $\mathcal{D}(x)$.

Proof. It is trivial to see that \preceq satisfies the properties of antisymetry, transitivity and totality required for a total order (see definition C.0.2 in appendix C). \square

Based on propositions 3.4.1 and 3.4.2, we define the natural split of a categorical variable as

$$A = \{x_i \in \mathcal{D}(x) \mid x_i \preceq \sigma\}$$

and

$$B = \{x_i \in \mathcal{D}(x) \mid x_i \not\preceq \sigma\}.$$

σ can logically be chosen from the possible categorical values of the variable x .

With this description of the possible splits based on a categorical variable, we conclude that every split can be perfectly defined by a variable (for ordered as well as for categorical variables) and a threshold value σ . This conclusion enables us to propose the next formal definition of decision trees, which is not just for ordered, but also for categorical variables.

⁵Since this proposition is the clue for the ordering of categorical variables, which allows us to develop a unique definition of DT for both categorical and ordered variables, we give it in this section before the introduction of this unique definition. However, for its perfect understanding, the reader needs to go to section 3.6.2.

⁶see definition 3.6.4

⁷see definition 3.6.3

⁸see definition 3.6.4

3.4.2 Formal definition of Decision Trees

For simplification of notation, from now on, we are considering that each variable, x_i for $i = 1, \dots, h$, of the measurement vector is of numerical type, i.e. $\mathcal{X} \subseteq \mathbb{R}^h$.⁹ Another assumption is that we will only consider binary trees. This consideration is based on the fact that every binary splitting procedure (each internal node has two child nodes) can always produce a multiway split tree (each internal node has more than two child nodes) (see section 3.4).

With this two assumptions, a split is perfectly defined by giving the variable and the threshold to make the division of the measurement space. Thereby, we give the following definition of decision tree:

Definition 3.4.5: Let $T = (V, E)$ be a rooted ordered binary tree, and let $V = V_1 \dot{\cup} \dots \dot{\cup} V_L$ be a partition where each $V_i, i = 1, \dots, L$ contains the vertices or nodes at each *level* of the tree (see definition 2.7.7 and theorem 2.7.2), i.e., for instance the elements of V_1 are the sources of the tree. A **decision tree** is composed of the tree T and a family of 2-tuples (q_i, σ_i) , each associated to one of the non leaves vertices $n_i \in V \setminus V_{\text{leaves}}$. The pair (q_i, σ_i) is called **split point** at node n_i .

For each node $n_i \in V \setminus V_{\text{leaves}}$ we have:

- $q_i \in \{1, \dots, h\}$, which represents the *feature axis*. Thus x_{q_i} is the variable used to split at node n_i ,
- $\sigma_i \in \mathcal{X}_{q_i}$ is the *threshold value* of the variable x_{q_i}

and for all $n_i \in V_{\text{leaves}}$, there exists a map

$$l : V_{\text{leaves}} \longrightarrow \mathcal{C}$$

which assigns a class label to each terminal node. In other words, each $n_i \in V_{\text{leaves}}$ has an element $l(n_i) \in \mathcal{C}$ associated.

Note: The knowledge represented in a decision tree can be extracted and represented in form of IF-THEN rules. One rule is created for each path from the root to a leaf node. Each *variable-threshold value* pair along a given path forms a conjunction in the rule antecedent (“IF” part). The leaf node holds the class prediction, forming the rule consequent (“THEN” part).

Graphical representation of definition 3.4.5

Figure 3.6 illustrates definition 3.4.5. In this figure, $T = (V, E)$ where $V = \{n_1, n_2, n_3, n_4, n_5\}$.

$V_{\text{leaves}} = \{n_3, n_4, n_5\}$ and the elements of $V \setminus V_{\text{leaves}} = \{n_1, n_2\}$ have the 2-tuplas $(2, \sigma_1)$ and $(1, \sigma_2)$ associated to n_1 and n_2 respectively. This 2-tuplas represent the partition of

⁹An order can be established on the categorical variables, and the split point of them can be calculated based on a threshold value (see previous section)

the measurement space shown in figure 3.8 and the comparison operators are illustrated in figure 3.7.

For this illustrative example, we consider the set of classes $\mathcal{C} = \{c_1, c_2, c_3\}$ and the class labels assigned to the leaf nodes are $l(n_3) = c_1$, $l(n_4) = c_2$ and $l(n_5) = c_3$.

Since $\#V_{leaves} = 3$, then 3 rules can be extracted from the decision tree of figure 3.6. For all $x = (x_1, x_2) \in \mathcal{X}$:

“IF $x_2 \leq \sigma_1 \wedge x_1 \leq \sigma_2$, THEN $class(x) = c_2$ ”

“IF $x_2 \leq \sigma_1 \wedge x_1 > \sigma_2$, THEN $class(x) = c_3$ ”

“IF $x_2 > \sigma_1$, THEN $class(x) = c_1$ ”

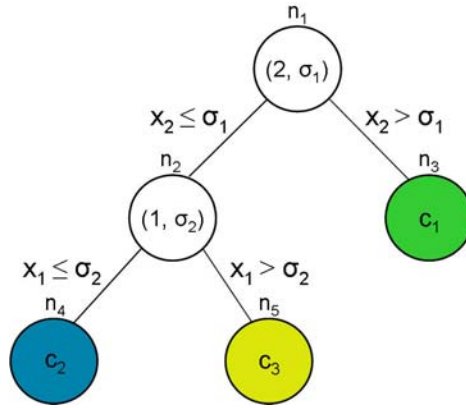


Figure 3.6: Example of a DT

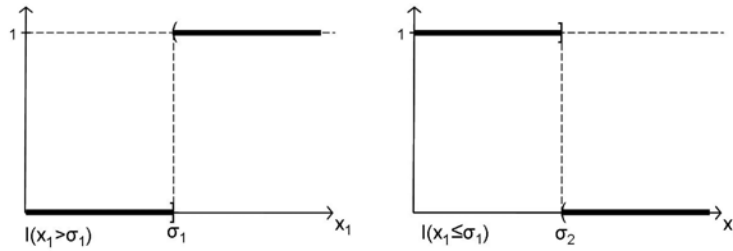


Figure 3.7: Greater and Less or equal operators of the previous example of a DT

A classification or decision tree partitions the measurement space \mathcal{X} into subregions corresponding to the terminal nodes, since each example will be classified by the label of the terminal node it reaches. Thus, decision trees can be seen as a hierarchical way to describe a partition of the measurement space.

In the next subsection we present a recursive way to define the partition of the measurement space made by a given decision tree, which is defined in the form of definition 3.4.5.

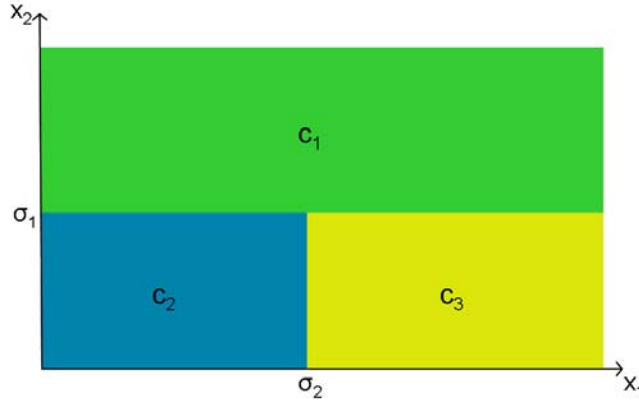


Figure 3.8: Space partitioning of the previous example of a DT

Partition of the measurement space for our definition of decision tree

Let T be a decision tree. For a measurement space \mathcal{X} and for each node $n_i \in V(T)$, we define a region R_i of \mathcal{X} in a recursive way as follows:

$$R_1 = \mathcal{X}$$

$$R_{2i} = \{x \in R_i \mid x \leq \sigma_i\}$$

and

$$R_{2i+1} = \{x \in R_i \mid x > \sigma_i\}$$

where (q_i, σ_i) is the 2-tuple associated to each $n_i \in V \setminus V_{leaves}$.

In other words, for each node n of a decision tree we define a subset R_n of the measurement space \mathcal{X} . The set R_n itself is a subset of the set $R_{P(n)}$ defined for the parent node $P(n)$ of n , and it comprises those elements of $R_{P(n)}$ which fulfill the condition imposed by the split point associated to the parent node.

Notation: From now on we will consider one measurement space \mathcal{X} that will be used to construct the decision tree. For simplification of notation, we will identify each node n_i with the region R_i associated to this node.

Applying tree based techniques for classification involves two subtasks: DT learning (training) to get the model and DT inference to classify instances. We summarize these processes in figure 3.9.

3.5 DT learning

Decision tree learning is also called discrimination and is the process of deriving rules from a given sample of classified objects $\mathcal{L} \subseteq \mathcal{X} \times \mathcal{C}$. The entire learning of the DT is divided into three tasks:

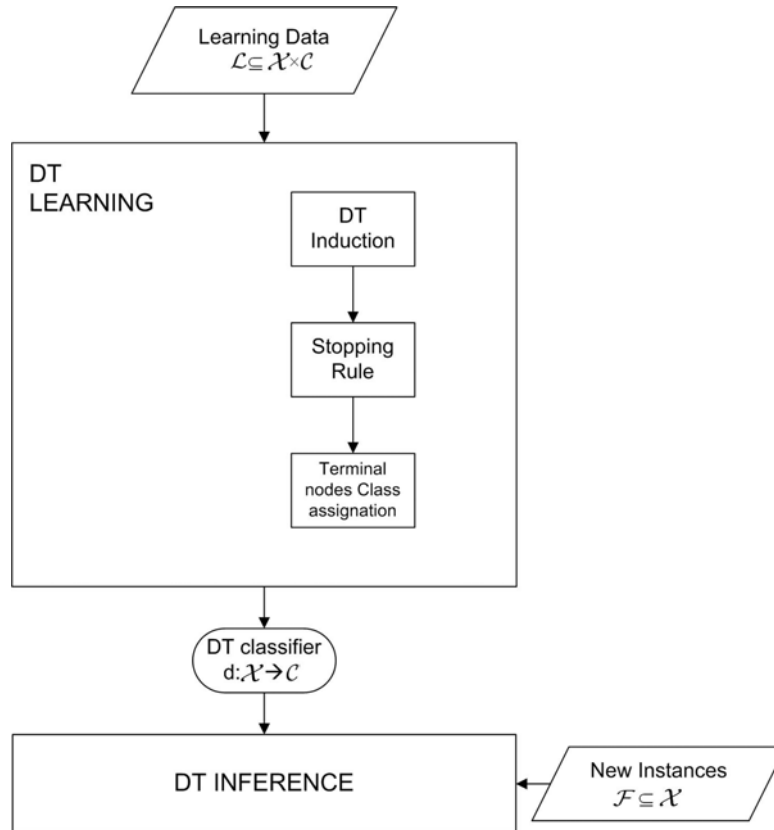


Figure 3.9: Outline of the learning and inference processes of a DT

1. The selection of the variable or feature axis q and threshold value σ of this variable q to split in every internal node. This task is called DT **induction**.
2. The decision when to declare a node terminal or to continue splitting it, which can be summarized in finding a stopping rule.
3. The assignation of a class to each of the terminal nodes.

3.6 DT induction

Decision tree (DT) induction is the task of constructing a tree from a learning sample \mathcal{L} . DTs are constructed recursively following a descendant strategy that begins at the root node. That is the reason why TDIDT (Top-Down Induction on Decision Trees) is the acronym used to make reference to the family of DT construction algorithms.

The tree is constructed by repeated splits of subsets of \mathcal{X} into two descendant subsets, beginning with \mathcal{X} itself. The fundamental idea is to select each node variable and threshold value, so that the data in each of the descendant subsets are “purer” w.r.t. class assignation than the data in the parent subset.

The question would be now: how can we measure how “pure” are the data in a certain node? There are functions, called impurity functions, which help us to measure this homogeneity of the data in relation to the class variable. So, if all the data in a set belong to the same class, then this set has zero impurity.

3.6.1 The standard n-simplex

For DT induction we need the concept of *simplex*, that will be useful to define a class of probability measures on each node of the tree.

Definition 3.6.1: For $k = 1, 2, 3, \dots$, the *standard simplex*¹⁰ Δ^k is defined as the set of all $u \in \mathbb{R}^k$ of the form

$$u = \sum_{i=1}^k \alpha_i \mathbf{e}_i$$

$$\alpha_i \geq 0 \quad \text{for } i = 1, \dots, k, \text{ and } \sum_{i=1}^k \alpha_i \leq 1,$$

¹⁰see [58]

where

$$\begin{aligned} \mathbf{e}_1 &= (1, 0, 0, \dots, 0), \\ \mathbf{e}_2 &= (0, 1, 0, \dots, 0), \\ &\vdots \\ \mathbf{e}_k &= (0, 0, 0, \dots, 1). \end{aligned}$$

is the standard basis of \mathbb{R}^k .

In another way, the standard k -simplex can be defined as the subset of \mathbb{R}^{k+1} given by

$$\Delta^k = \{(t_0, \dots, t_k) \in \mathbb{R}^{k+1} \mid \sum_i t_i = 1 \text{ and } t_i \geq 0 \text{ for all } i\}.$$

Removing the restriction $t_i \geq 0$ in the above gives an k -dimensional affine subspace of \mathbb{R}^{k+1} containing the standard k -simplex. The vertices of the standard k -simplex are the points of the standard basis of \mathbb{R}^{k+1} .

There is a canonical map from the standard k -simplex to an arbitrary k -simplex with vertices (v_0, \dots, v_k) given by

$$(t_0, \dots, t_k) \mapsto \sum_i t_i v_i$$

The coefficients t_i are called the *barycentric coordinates* of a point in the k -simplex. Such a general simplex is often called an *affine k -simplex*, to emphasize that the canonical map is an *affine transformation*. It is also sometimes called an *oriented affine k -simplex* to emphasize that the canonical map may be *orientation preserving* or *reversing*.

3.6.2 The splitting Rule

Algorithms for building decision trees at the same time try to optimize the performance of the tree and to create a tree as small as possible. This is done by selecting attributes to be included into the tree according to some information theoretical measure.

To build a decision tree it is necessary to find at each internal node a split for the data. The goodness of split criterion was originally derived from an impurity function.

Definition 3.6.2: An **impurity function** is a function $\Phi : \Delta^{(J-1)} \longrightarrow \mathbb{R}$ defined on the standard (topological) $(J-1)$ -simplex ¹¹

$$\Delta^{(J-1)} = \{(p_1, \dots, p_J) \in \mathbb{R}^J \mid \sum_i p_i = 1 \text{ and } p_i \geq 0 \text{ for all } i\}$$

with the properties

1. Φ has a unique maximum at the point $(\frac{1}{J}, \frac{1}{J}, \dots, \frac{1}{J})$,

¹¹ $J \in \{1, 2, \dots\}$. In our classification context J will be the number of possible classes

2. Φ achieves its minimum only at the vertices of the standard $(J-1)$ -simplex, i.e., at the components of the standard basis of \mathbb{R}^J $\{\mathbf{e}_1, \dots, \mathbf{e}_J\}$, $(1, 0, \dots, 0)$, $(0, 1, \dots, 0)$, \dots , $(0, 0, \dots, 1)$,
3. Φ is a symmetric function on J variables p_1, \dots, p_J , i.e., is a function that is unchanged by any permutation of its variables.

Definition 3.6.3: Given a learning sample $\mathcal{L} = (X, C) \subseteq \mathcal{X} \times \mathcal{C}$ and a node $n \in V$, define the **node n proportions** $p(j|n)$, $j \in \mathcal{C} = \{1, \dots, J\}$ to be the proportion of the cases $x \in R_n \cap X$ (see section 3.4.2) belonging to class j , so that

$$p(1|n) + \dots + p(J|n) = 1.$$

If $\mathcal{X} \subseteq \mathbb{R}^h$, then the node proportions are elements of the standard simplex Δ^{h-1} . Normally, this proportion is defined as the frequency of class j in node n :

$$p(j|n) = \frac{N_j(n)}{N(n)}$$

being $N_j(n)$ the number of class j cases of X in n , i.e.,

$$N_j(n) = \sum_{x^{(i)} \in R_n \cap X} I(y^{(i)} = j)$$

where $I(\cdot)$ is the indicator function and $N(n)$ the total number cases in X belonging to n ,

$$N(n) = \#\{x \in R_n \cap X\}$$

Definition 3.6.4: Given an impurity function $\Phi : \Delta^{(J-1)} \longrightarrow \mathbb{R}$, the **impurity measure** $i : \text{Nodes}(T) \longrightarrow \mathbb{R}$ of any node n is defined as

$$i(n) = \Phi(p(1|n), p(2|n), \dots, p(J|n)). \quad (3.3)$$

Suppose that a split candidate $s = (q, \sigma)$ divides the data at node n into two subsets such that the proportion p_L goes to the left child node n_L and the proportion p_R goes to the right child node n_R . The **goodness of the split** is defined as the difference of the weighted average impurity

$$\Delta i(s, n) = i(n) - p_R i(n_R) - p_L i(n_L). \quad (3.4)$$

Proposition 3.6.1: Let T be a K -ary decision tree (for binary trees $K = 2$) and let $i : \text{Nodes}(T) \longrightarrow \mathbb{R}$ be the impurity measure defined by a strictly concave impurity function $\phi : \Delta^{(J-1)} \longrightarrow \mathbb{R}$, where J is the number of classes of our classification problem, i.e.,

$$i(n) = \phi(p(1|n), p(2|n), \dots, p(J|n)). \quad (3.5)$$

for all $n \in \text{Nodes}(T)$. Then the difference of the weighted average impurity is non-negative, and zero if and only if the distributions are the same in all children, i.e.,

$$\Delta i(s, n) \geq 0$$

and

$$\Delta i(s, n) = 0 \quad \text{if and only if} \quad i(n) = i(n_k), \quad \text{being } n_1, \dots, n_K \text{ a partition of } n$$

.

Proof. (see [10])

$$\Delta i(s, n) = i(n) - \sum_{k=1}^K p_k i(n_k)$$

being K the number of children nodes of $n \in V(T)$, and then $\bigcup_{k=1}^K n_k = n$ and $n_k \cap n_j = \emptyset$ for all $k \neq j$, $k, j \in 1, \dots, K$

$$\sum_{k=1}^K p_k i(n_k) = \sum_{k=1}^K p_k \phi(p(1|n_k), \dots, p(J|n_k))$$

and by Jensen's inequality and the concavity of ϕ it is fulfilled that

$$\sum_{k=1}^K p_k \phi(p(1|n_k), \dots, p(J|n_k)) \leq \phi\left(\sum_{k=1}^K p_k (p(1|n_k), \dots, p(J|n_k))\right).$$

If we apply the theorem of total probability, then, since $\{n_1, \dots, n_K\}$ is a partition of n ,

$$\sum_{k=1}^K p_k (p(1|n_k), \dots, p(J|n_k)) = (p(1|n), \dots, p(J|n))$$

and thus, we have

$$\sum_{k=1}^K p_k i(n_k) \leq \phi(p(1|n), \dots, p(J|n))$$

which is equal to the impurity of node n , $i(n)$. □

In our case, the tree is always binary, and we can express this theorem in terms of right and left child nodes in the following way:

$$i(n) - p_L i(n_L) - p_R i(n_R) \geq 0.$$

Remark: The definition of impurity function does not imply that ϕ is strictly concave. This is a further condition, which, together with the third condition of definition 3.6.2, does imply that ϕ reaches its maximum at the point $(\frac{1}{J}, \dots, \frac{1}{J})$, which is the first condition of the definition.

12

In the next sections we will present the most used types of impurity measures:

- Gini index of diversity
- Information Gain
- Misclassification rate

3.7 Gini index of diversity

Adopted by Breiman et al. in their work [10], the Gini index of diversity (Corrado Gini 1884-1965) is based on the squared probabilities of membership for each element of \mathcal{C} to node n . The Gini impurity function has the following form:

$$\Phi_G : \Delta^{(J-1)} \longrightarrow \mathbb{R}$$

$$\Phi_G(\pi_1, \pi_2, \dots, \pi_n) = \sum_{k=1}^J \sum_{\substack{j=1 \\ j \neq k}}^J \pi_j \pi_k$$

and thus, the Gini impurity measure of a node n would be:

$$i_G(n) = \Phi_G(p(1|n), p(2|n), \dots, p(J|n)) \quad (3.6)$$

$$= \sum_{k=1}^J \sum_{j=1, j \neq k}^J p(j|n)p(k|n), \quad (3.7)$$

for $p(j|n)$ defined in 3.6.3.

¹²By Jensen's inequality it is fulfilled that

$$\sum p_i i(p(c | a_i)) \leq i\left(\sum p_i p(c | a_i)\right) = i(p_c)$$

with equality if and only if $p(a | a_i) = p(c)$ for all i and c .

The Gini impurity measure is normally called *Gini index of diversity* and can be written as:

$$i_G(n) = \sum_{j=1}^J p(j|n) - \sum_{j=1}^J (p(j|n))^2 \quad (3.8)$$

$$= 1 - \sum_{j=1}^J (p(j|n))^2 \quad (3.9)$$

Interpretation

As it is explained in [55] and [10], there are two main interpretations of the Gini index:

- First, the interpretation of the Gini index as the expected error rate if the class label is randomly chosen from the class distribution at the node. In other words, use the rule that assigns an object selected at random from the node n to class k , $k \in \{1, \dots, J\}$ with probability $p(k|n)$. $p(j|n)$ will be the estimated probability that the selected object class is actually $j \in \{1, \dots, J\}$. Thereby, the estimated probability of misclassification under this assignation rule is $\sum_{j \neq k} p(j|n)p(k|n)$, which coincides with the definition of Gini index.
- A further interpretation of the Gini index is in terms of variances. In a node n , assign all class $j \in \{1, \dots, J\}$ elements the value 1, and assign the value 0 to all other elements of the node. Thus, the sample variance of these values is $p(j|n)(1 - p(j|n))$. If this is repeated for all classes in $\{1, \dots, J\}$ and the variances are summed, the result is

$$\sum_{j=1}^J p(j|n)(1 - p(j|n)) = 1 - \sum_{j=1}^J (p(j|n))^2.$$

Graphical representation

If we have a two-class problem, the Gini index of diversity is

$$\begin{aligned} i_G(n) &= \Phi_G(p(1|n), p(2|n)) \\ &= \Phi_G(p(1|n), 1 - p(1|n)) \end{aligned}$$

since $(p(1|n), p(2|n))$ are elements of the simplex Δ^1 , then

$$\begin{aligned} i_G(n) &= \sum_{k=1}^2 \sum_{j=1, j \neq k}^J p(j|n)p(k|n) \\ &= p(1|n)(1 - p(1|n)) + (1 - p(1|n))p(1|n) \\ &= 2(p(1|n) - (p(1|n))^2) \end{aligned}$$

and this formula is represented in figure 3.10, where the probabilities $p(1|n)$ are represented at the x -axis and the $i_G(n) = 2\left(p(1|n) - (p(1|n))^2\right)$ at the y -axis.

Remark: The Gini impurity function $\Phi_G : \Delta^{J-1} \rightarrow \mathbb{R}$ is a quadratic polynomial with non-negative coefficients and hence it is strictly concave. This can be observed in its representation for the two-class problem in figure 3.10 .

Then, according to proposition 3.6.1, for any split s chosen at node n , the goodness of s (see definition 3.6.4) is always positive

$$\Delta i_G(s, n) \geq 0.$$

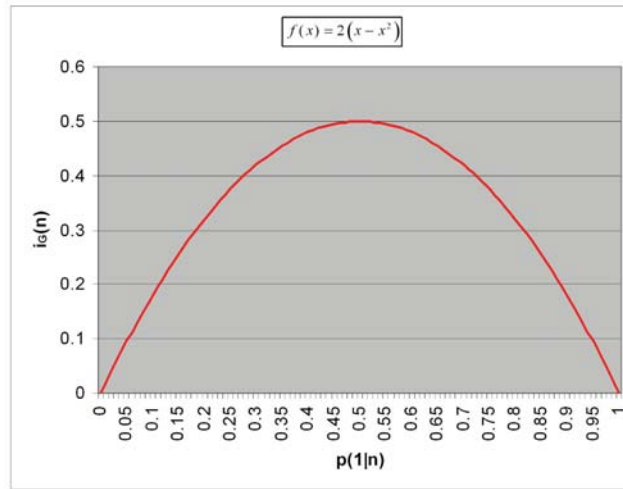


Figure 3.10: Graphical representation of the Gini Index of diversity for a two-class classification problem

3.8 Information Gain impurity measure

Mainly used by Quinlan in his work [60], the information Gain function is based on the notion of entropy, which characterizes the impurity of an arbitrary set of examples. It has the following form:

$$i_{Gain}(n) = - \sum_{j=1}^J p(j|n) \log_2 p(j|n) \quad (3.10)$$

Interpretation

The information Gain function has its origin in information theory and it can be interpreted in the following way:

If we randomly select an example from a set n and we assign this example to class c_j , $j \in \{1, \dots, J\}$, then $p(c_j|n) = \frac{N_j(n)}{\#n}$, where $N_j(n)$ denotes the number of elements of n belonging to class c_j , is the probability of this message and the amount of information it conveys is $-\log(p(c_j|n))$ ¹³. The expected information provided by the message with respect to the class membership would be equal to $-\sum_{j=1}^J p(c_j|n) \log_2 p(c_j|n)$.

The value $i_{Gain}(n)$ measures the average amount of information needed to identify the class of an example in n .

The logarithm is in base 2 because the entropy is a measure of the expected encoding length measured in bits.

¹⁴

Graphical representation

In the two class problem, the Information Gain index of a set n is¹⁵

$$\begin{aligned} i_{Gain}(n) &= -p(1|n) \log_2 p(1|n) - p(2|n) \log_2 p(2|n) \\ &= -p(1|n) \log_2 p(1|n) - (1 - p(1|n)) \log_2 (1 - p(1|n)) \end{aligned}$$

and it is represented in figure 3.11

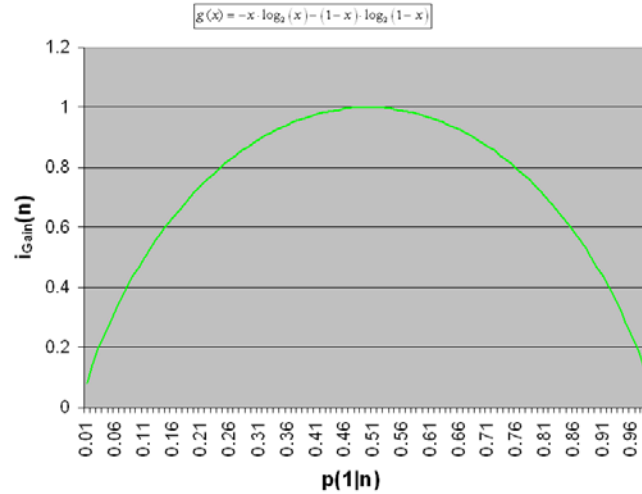


Figure 3.11: Graphical representation of the Gain impurity function for a two-class classification problem

¹³ \log in this case refers to \log_2

¹⁴The $\log_2 : [0, +\infty] \rightarrow \mathbb{R}$ function multiplied by the identity function $i : \mathbb{R} \rightarrow \mathbb{R}$ and restricted to the domain interval $[0, 1]$ gives as result the function $g : [0, 1] \rightarrow \mathbb{R}$, $g(x) = x \log_2(x)$, which is convex. The function $-g(x)$ is then concave and fulfills the requirement of the impurity function definition 3.6.2.

¹⁵ $p(1|n)$, $p(2|n)$ are elements of the simplex

3.9 Misclassification rate impurity measure

It is a very natural and intuitive impurity measure since it is based on the notion of reduction of the misclassification rate (see [70], [10]). The impurity function for this criteria has the form

$$\begin{aligned}\Phi_{MR} : \Delta^{J-1} &\longrightarrow \mathbb{R} \\ \Phi_{MR}(p_1, \dots, p_J) &= 1 - \max_{j \in \{1, \dots, J\}} p_j\end{aligned}\quad (3.11)$$

This function has all desirable properties listed in definition 3.6.2.¹⁶

The origin of this formula is to take the tree impurity as $R(T)$, the resubstitution estimate for the expected misclassification cost. The split chosen would be then the one that leads to the highest reduction of the estimated misclassification rate $r(t)$ for any node t , where

$$\begin{aligned}r(t) &= \min_i \sum_{j=1}^J C(i|j)p(j|t) \\ &= 1 - \max_j p(j|t) \text{ in the unit cost case.}\end{aligned}$$

$C(i|j)$ is the cost of misclassifying an object as class i if it belongs to class j . The equivalent impurity function is then the previously described Φ_M (3.11) and its representation for the two-class problem can be seen in figure 3.12

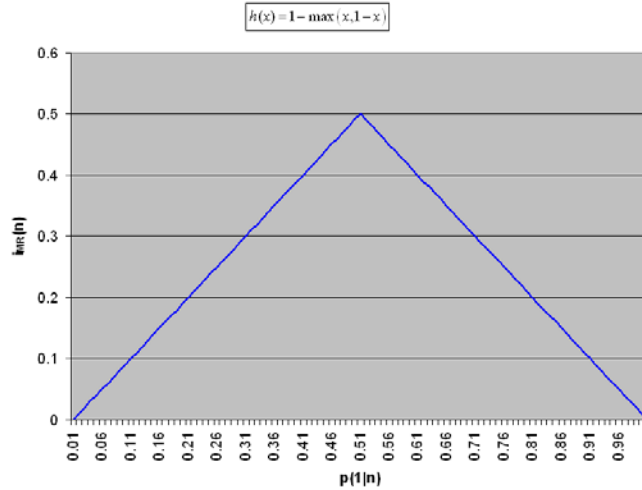


Figure 3.12: Graphical representation of the Misclassification rate impurity function for a two-class classification problem

¹⁶The origin of this function is the search of an impurity function $\Phi : \Delta^{J-1} \longrightarrow [0, +\infty)$ having the properties of definition 3.6.2.

3.10 Gini versus Gain

Several authors have tried to compare the behaviour of the two most popular split functions, namely the Gini Index of diversity and the Information Gain without finding a guide to determine which one performs better in each case. In [53], for example, they conclude that it is not possible to decide which one performs better, since they disagree only in 2% of all cases. In figure 3.13 the graphical representation of this two impurity measures and of the Misclassification rate impurity measure for a 2-class classification problem are presented together. In this figure, the x-axis represents the estimated probability that a object selected in node n is actually of class 1, i.e., $p(1|n)$ and the y-axis represents the impurity of node n measured with these three different impurity measures.

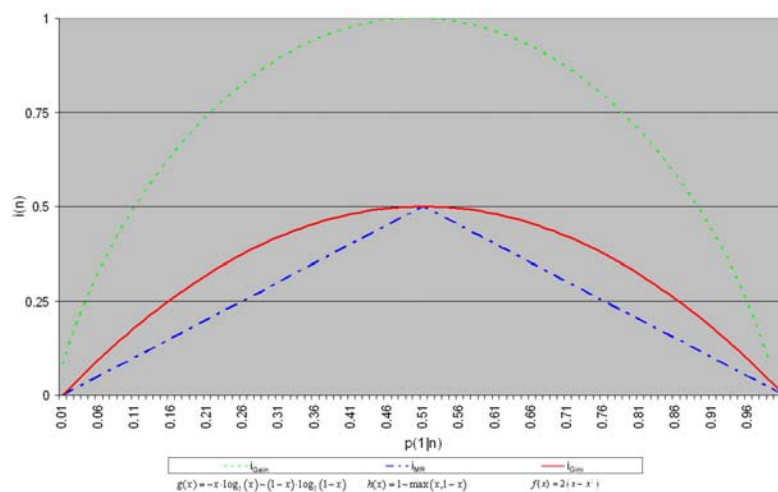


Figure 3.13: Impurity functions for a two class classification problem

3.11 Variable combinations

The splits described in section 3.4.1 are based on single attributes. One of the obvious limitations of univariate trees is that their internal nodes can only separate the data with hyperplanes parallel to the coordinate axes. Some classification problems have a structure that suggest treatment through combination of variables. Multivariate decision trees attempt to generate decision trees by employing discriminant functions at internal nodes with more than one variable. In the particular case of linear decision trees, for example, the partition of the measurement space is done with hyperplanes of arbitrary slope, rather than only parallel to the axes.

¹⁷ ¹⁸

¹⁷An interesting method to find parameters for the oblique decision tree can be seen in [64]

¹⁸The optimal split at every internal node will be, as well as by univariate splits, the one which maximizes the decrease of impurity function

Algebraic decision trees

Another view of multivariate splits is given in [20] and they are called linear decision trees, or more general, algebraic decision trees ([34], [19]) :

Linear Decision Trees

In a linear decision tree (with input size h) every internal node is labeled with a vector (a_0, a_1, \dots, a_h) and has three outgoing edges labeled $-$, 0 , and $+$. Given an input vector (x_1, x_2, \dots, x_h) , we decide which way to branch based on the sign of the following expression:

$$a_0 + a_1x_1 + a_2x_2 + \dots + a_hx_h.$$

For example, in each node in a comparison tree ¹⁹, we have $a_i = 1$ and $a_j = -1$ for some i and j , and $a_k = 0$ for all $k \neq i, j$.

Linear decision trees have a very simple geometric interpretation. A generic 2-class problem can be thought of as a function $F : \mathbb{R}^h \rightarrow \{0, 1\}$, where the input is a point in h -space and the output is a single bit. Every internal node defines an hyperplane with equation:

$$a_1x_1 + a_2x_2 + \dots + a_hx_h = -a_0,$$

where $a = (a_0, a_1, a_2, \dots, a_h)$ is thus node-dependent. This equation describes a line when $h = 2$, a plane when $h = 3$, and so on. We branch at a node depending on whether an input is above, below, or on this hyperplane.

Now consider the set of input points $R_v \subseteq \mathbb{R}^h$ that reach a particular node v in a linear decision tree (see analogous in section 3.4.2). The set R_v contains all the points that satisfy a set of linear equalities and inequalities; such a set is called a convex polyhedron.

Algebraic Decision Trees

Now let us consider the following obvious generalization of linear decision trees, first proposed by Guy Steele and Andy Yao ([34]). In a d th-order algebraic decision tree, every node v is labeled with a polynomial $q_v \in \mathbb{R}[x_1, x_2, \dots, x_h]$ of degree at most d . As in the linear case, each node has three branches labeled $-$, 0 , and $+$, and the computation at node v branches according to the sign of the polynomial expression $q_v(x)$. A first order decision tree is just a linear decision tree.

3.12 Stopping Rule

A further component of the decision tree learning process is to determine when to stop splitting. Thus, a stopping rule is needed. Several approaches can be found in the literature and we present some of them:

¹⁹see [21], a comparison tree is a tree where each node is labeled with two indices $1 \leq i, j, \leq h$ and two edges labeled $<$ and $>$, corresponding to the two cases $x_i < x_j$ and $x_i > x_j$.

- Thresholds on impurity: In this approach, a node n is declared terminal if a split on n would not improve significantly the impurity measure. It is done in the following way:

- Set a threshold $\beta > 0$
- Declare a node n terminal if

$$\max_{s \in S} \Delta i(s, n) < \beta,$$

where S is the set of possible splits on n .

This rule produces generally unsatisfactory results. The problem is that if β is set too low, then the size of the tree gets too large. Another aspect of this problem is that if β is increased, then there may be nodes n such that $\Delta i(s, n) < \beta$ for all possible split $s \in S$ of n , but the descendant nodes n_L, n_R of n may have splits s_L or s_R for example, so that $\Delta i(s_L, n_L) \gg \beta$ or $\Delta i(s_R, n_R) \gg \beta$. By declaring n terminal, the splits s_L or s_R are lost.

- Threshold on the node size: This approach declares a node n terminal if its number of elements is smaller than a certain $k \in \mathbb{N}$. This strategy is known to be not robust.

With the previous notation, this method would follow this rule: For all node n , if $\#R_n < k$ then stop splitting.

- Two stage search: This approach divides tree induction in two subtasks:
 - Determine the structure of the tree, i.e., determine a tree $T = (V, E)$ where the nodes are labelled with natural numbers
 - Find splits for all the nodes, i.e., find the associated 2-tuple (q, σ) for each node $n \in V$.

The optimization in the first stage may or may not be related to that used in the second stage. An easy approach is to fix, normally based on a priori information, the height of the tree in the first stage and use impurity measures to determine the associated 2-tuples for each node in the second stage. Alternative approaches not using impurity functions are for example the one used by Lin and Fu ([67]), who uses K-means clustering for both stages, or the one used by Quin Yun and Fu ([63]), who use multi-variate stepwise regression for the first stage and linear discriminant analysis for the second stage.

Breiman *et al* conclude that, after testing some methods, looking for the right stopping rule is the wrong way of looking at the problem of finding appropriate sized decision trees. They suggest, instead of using stopping rules, the construction of a tree until all leaf nodes impurity is nearly zero, obtaining thereby a large tree. Then, selectively, remove subtrees that are not contributing significantly towards generalization accuracy. These methods are called *Pruning* methods and we give an overview of them in section 3.15.

3.13 Terminal nodes class assignation

The class assignment process is the easiest. After each splitting procedure, classes are assigned to the terminal nodes using the majority rule. In order to minimize the misclassification rate, each terminal node is assigned to the class with the highest probability in this node. These probabilities are usually computed as the coefficient of the number of samples belonging to each class at that specific terminal node and the cardinality of this node. The function

$$l : V_{leaves} \longrightarrow \mathcal{C}$$

(see definition 3.4.5) that assigns a class label to each terminal node n , denoted by **class assignment rule**, is then

$$l(n) = \arg \max_{j \in \mathcal{C}} p(j|n),$$

where $p(j|n) = \frac{N_j(n)}{N(n)}$, being $N_j(n)$ the number of class j objects in n and $N(n)$ the total number objects in n .

If the maximum is achieved for more than one element in \mathcal{C} , then $l(n)$ is arbitrarily assigned as one of the maximizing classes.

3.14 Accuracy

Let (X, Y) be jointly distributed random variables with h -dimensional vector X denoting a feature or measurement vector that takes values from \mathcal{X} and Y denoting the associated class label of X . Y takes values from $\mathcal{C} = \{1, \dots, J\}$. Then, a classifier $d(\cdot)$ will estimate Y based on observing X .

Definition 3.14.1: Let $d : \mathcal{X} \longrightarrow \mathcal{C}$ be a classifier, The **true misclassification rate** of d , denoted by $R^*(d)$, is

$$R^*(d) = P(d(X) \neq Y).$$

where P denotes the probability.

Conceptually, the process of obtaining the true misclassification rate $R^*(d)$ of a classifier created from a learning data set would be:

- Construct $d(\cdot)$ using \mathcal{L} .
- Draw a virtually infinite set of cases from the same population as \mathcal{L} was drawn from.
- Observe the correct classification for each of these cases.
- Find the predicted classification using $d(\cdot)$.
- $R^*(d)$ is the proportion of misclassifications by d .

To make the previous concept precise, we need a probability model:

- Define the space of the cartesian product of the measurement space and the set of classes, $\mathcal{X} \times \mathcal{C}$.
- Let $P(A, j)$ be a probability on $\mathcal{X} \times \mathcal{C}$, $A \subset \mathcal{X}$, $j \in \mathcal{C}$
- Assume that the learning sample $\mathcal{L} \subset \mathcal{X} \times \mathcal{C}$ consists of N objects $(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})$ independently drawn randomly from the distribution $P(A, j)$.
- Construct the classification rule $d(\cdot)$ based on \mathcal{L} .
- $R^*(d)$ is the probability that d will misclassify a new sample coming from the same distribution as \mathcal{L} .

With this probability model, the definition of $R^*(d)$ holds:

Definition 3.14.2: Let \mathcal{L} be a learning example, following a probability distribution $P(A, j)$, and let $d(\cdot)$ be a classifier obtained by a learning process of \mathcal{L} . Let (X, Y) , $X \subseteq \mathcal{X}$, $Y \in \mathcal{C}$ be a new sample from the probability distribution $P(A, j)$, it means,

1. $P(X \in A, Y = j) = P(A, j)$
2. (X, Y) and \mathcal{L} are independent

Then, $R^*(d) = P(d(X) \neq Y)$.

There is no difficulty to calculate $R^*(d)$ if the learning sample \mathcal{L} is composed of simulated data following a certain distribution function. But in most of the cases only the data in \mathcal{L} are available with no possibility of getting an additional large sample of classified measurement vectors. Thus, due to the difficulty of computing the true misclassification rate, it is estimated and \mathcal{L} must be applied both to construct the function $d(\cdot)$ and to estimate $R^*(d)$. These estimates of $R^*(d)$ are called **internal estimates**. The estimation of the true misclassification rate can be done in several ways:

Resubstitution estimate

It is the most commonly used estimate for the true misclassification rate.

Definition 3.14.3: Let $\mathcal{L} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ be a learning sample, where $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{C}$. Let a classifier $d : \mathcal{X} \rightarrow \mathcal{C}$ constructed from \mathcal{L} . The resubstitution estimate, denoted by $R(d)$ is given by

$$R(d) = \frac{1}{N} \sum_{i=1}^N I(d(x^{(i)}) \neq y^{(i)})$$

where $I(\cdot)$ is the indicator function.

Resubstitution estimate is thus computed using \mathcal{L} , which is the same data sample used to construct d . The fact that the two data sets are not independent can derivate in overfitting, which means that the classifier is too much adjusted to the learning data sample. This possible overfitting of the classifier to the learning data is due to the fact that all classification procedures try to minimize, either directly or indirectly, $R^*(d)$ or its estimate.

Test sample estimate

In this method, the available set of objects is divided in two parts: the **training** data sample $\mathcal{L}^{(1)}$ and the **test** data sample $\mathcal{L}^{(2)}$. $\mathcal{L}^{(1)}$ is used to construct the classifier d and $\mathcal{L}^{(2)}$ to estimate $R^*(d)$. Commonly, $\mathcal{L}^{(1)}$ is taken to be randomly sampled 2/3 of \mathcal{L} and $\mathcal{L}^{(2)}$ the remaining 1/3 of \mathcal{L} . Thereby, if N_1 is the number of training objects and N_2 the number of test objects, then

$$N_1 = \frac{2N}{3} \quad \text{and} \quad N_2 = \frac{N}{3}$$

Definition 3.14.4: Let $\mathcal{L} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ be a learning sample and let $\mathcal{L} = \mathcal{L}^{(1)} \cup \mathcal{L}^{(2)}$ be a bipartition of \mathcal{L} . If $d : \mathcal{X} \rightarrow \mathcal{C}$ is a classifier constructed from \mathcal{L} , then the test sample estimate, denoted by R^{ts} , is

$$R^{ts}(d) = \frac{1}{N_2} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{L}^{(2)}} I(d(x^{(i)}) \neq y^{(i)}).$$

This test sample approach reduces the number of objects used to construct the classifier. This reduction can be a drawback if the sample size $\#\mathcal{L}$ is small. To solve this drawback, the V-fold crossvalidation method is applied.

V-fold cross-validation

In this method, the available set of samples \mathcal{L} is randomly divided into V subsets of nearly equal size, $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_V$ with $N_v = \#\mathcal{L}_v$. For every $v \in \{1, \dots, V\}$, the test sample estimate is applied for a learning sample $\mathcal{L}^{(1)} = \mathcal{L} \setminus \mathcal{L}_v$ and a training sample \mathcal{L}_v . Let $d^{(v)}(\cdot)$ be the resulting classifier, the test sample estimate for $R^*(d^{(v)})$ is given by

$$R^{ts}(d^{(v)}) = \frac{1}{N_v} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{L}_v} I(d^{(v)}(x^{(i)}) \neq y^{(i)}).$$

We will have then a classifier $d^{(v)}(\cdot)$ and a test sample estimate $R^{ts}(d^{(v)})$ for each $v = 1, \dots, V$. The classifier d is constructed using \mathcal{L} and the V-fold cross-validation estimate, denoted by $R^{cv}(d)$, is given by

$$R^{cv}(d) = \frac{1}{V} \sum_{v=1}^V R^{ts}(d^{(v)}).$$

Every object in \mathcal{L} is used for the construction of the classifier and each of them is used once in a test sample. Notice that when $V = N = \#\mathcal{L}$, N-fold cross-validation is also called **leave-one-out** estimate. In tree structured classifiers tenfold cross-validation is usually applied.

3.15 Pruning

Pruning is the method most widely used for obtaining right sized trees. It was proposed by Breiman *et al* in their work [10] and consists of building the complete tree and then remove subtrees that are not contributing significantly towards generalization accuracy. The argumentation why this method will lead to more accurate decision trees than stopping rules is because it can compensate for the suboptimality of greedy tree induction. For a more detailed description of pruning, we need some previous definitions:

Definition 3.15.1: For the class assignation rule $l(n)$ described in the previous section 3.13, the resubstitution estimate $r(n)$ of the probability of misclassification, given that an object falls into node n , is

$$r(n) = \sum_{j \neq l(n)} p(j|n) = 1 - p(l(n)|n) = 1 - \max_{j \in \mathcal{C}} p(j|n).$$

Denote $R(n) = r(n)p(n)$, where $p(n)$ is the probability of a random object falling into node n . $R(n)$ is called **node misclassification cost**.

Then, the resubstitution estimate for the overall misclassification rate $R^*(T)$ of the tree classifier $T = (V, E)$, denoted by **tree misclassification cost** is given by

$$R(T) = \sum_{n \in V_{leaves}} r(n)p(n) = \sum_{n \in V_{leaves}} R(n)$$

Definition 3.15.2: Let $T = (V, E)$ be a tree and let T_t be a branch of T with root node $t \in V$ (see definition 2.7.4). **Pruning** the branch T_t from T consists of deleting from T the principal subtree (see definition 2.7.12) of T except t itself, i.e. all descendants of t . The resulting **pruned tree** is denoted by $T - T_t$.

Example As an example of a pruned tree, see figure 3.14

Definition 3.15.3: Let T be a tree and let T' be a tree obtained by successively pruning of branches from T , then T' is denoted by $T' \prec T$ and T' is called a **pruned subtree** of T

The number of rooted subtrees of a binary tree is very large, so a way to deal with this family is needed.

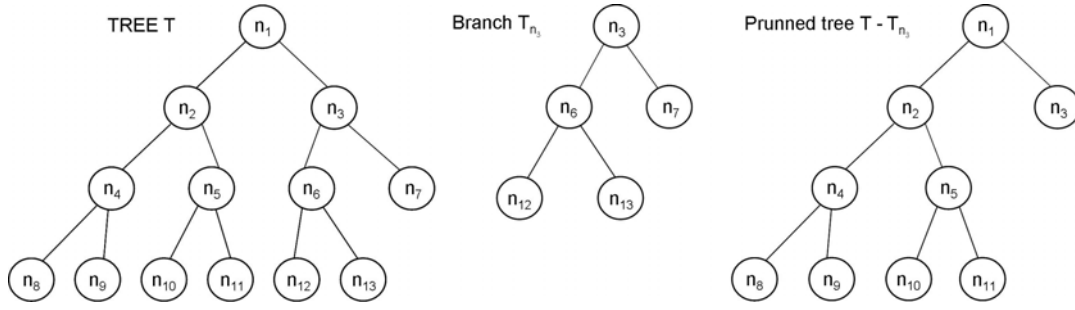


Figure 3.14: Example pruned decision tree

Minimal cost-complexity pruning

Definition 3.15.4: For any $T \preceq T_{max}$, the **complexity** of T is defined as its number of leaf nodes. If we denote by \tilde{T} the set of leaf nodes of T ($V_{leaves}(T)$), then the complexity of T is $\#\tilde{T}$. Let $\alpha \geq 0$, $\alpha \in \mathbb{R}$ be a parameter called the **complexity parameter** and the **cost-complexity measure**, denoted by $R_\alpha(T)$, is defined as

$$R_\alpha(T) = R(T) + \alpha \#\tilde{T}.$$

α can be interpreted as the complexity cost per terminal node and $R_\alpha(T)$ is then formed by adding to the misclassification cost of the tree a cost penalty for complexity. For each value of α , we will have a subtree $T(\alpha) \preceq T_{max}$ which minimizes $R_\alpha(T)$.

Definition 3.15.5: The smallest minimizing subtree $T(\alpha)$ for each complexity parameter α is defined by the conditions:

- $R_\alpha(T(\alpha)) = \min_{T \preceq T_{max}} R_\alpha(T)$
- If $R_\alpha(T) = R_\alpha(T(\alpha))$, then $T(\alpha) \preceq T$.

The aim of pruning is, for all α , to find the smallest minimizing subtree $T(\alpha) \preceq T_{max}$.

If α is very small, then T_{max} is the optimal one. On the contrary, if α increases much, then it remains just the root node.

If we have $T_1, T_2, \dots, root$, where T_i is the optimal subtree for $\alpha = \alpha_i$, then among these trees, the tree that provides the smallest test sample or cross-validation estimate of the misclassification rate, is selected.

3.16 DT inference

Decision tree inference or classification is the application of the rules to new objects of unknown class, i.e., classification is applying the rules to determine the class of new objects of unknown class. Our aim in this section is to propose a function which describes the intrinsically defined rules in a DT. This function would be the classifier (see definition 3.3.5)

represented by the DT. Due to the best of our knowlegde, the classifier function has never been described in an explicit way in the literature.

Description of the problem: From tree to classification rule Let T be a decision tree in the form of definition 3.4.5 and let $x \in \mathcal{X}$ be a measurement vector to be classified, which is given in form of h -tuples whose classification is unknown. Our objective is to define the classifier or classification rule (see definition 3.3.5) $d_T : \mathcal{X} \rightarrow \mathcal{C}$ which T represents and that will assign a class label to each measurement vector.

First of all, for a given measurement vector $x \in \mathcal{X}$ and a given DT T , we need to know to which leaf n^* of T does x arrive after passing through the tree, and then assign to x the class label of n^* (see section 3.13). To determine which leaf node does x reach, a test is done to x at each internal node $n_i \in V(T)$ along the path that begins at the root node and finish at a leaf node $n^* \in V_{\text{leaves}}(T)$. At each internal node n_i with 2-tuple (q_i, σ_i) associated (see definition 3.4.5), the test will determine if x goes to the right or to the left child node of n_i , i.e., if x goes to n_{2i} or to n_{2i+1} . We propose the following definition of test:

Definition 3.16.1: Let T be a DT and let $\mathcal{X} \subseteq \mathbb{R}^h$ be a measurement space. For all $x \in \mathcal{X}$ and for all $i \in \{j | n_j \in V(T)\}$, we define a **test** to x at node n_i as the evaluation of a function $f : \mathcal{X} \times \mathbb{N} \rightarrow \{0, 1\}$

$$f(x, i) = I(x_{q_i} > \sigma_i),$$

where I is the indicator function.

A test is done to $x \in \mathcal{X}$ at one node at each level of the tree T to determine to which node of the next level does x go. If $f(x, i) = 0$, then x goes to the left child node of n_i , i.e., to node n_{2i} and if $f(x, i) = 1$, then x goes to the right child node of n_i , which is n_{2i+1} . This process begins at the root node of T and ends at a terminal node. At the end of this process we will have a vector $G \in \bigcup_{N \in \mathbb{N}} \{0, 1\}^N$ whose components G_j will be the result of the test function f applied to x at exactly one node at each internal level j of T and at the root node. Moreover, if L is the level of the terminal node x reaches, then the dimension of G will be $L - 1$, i.e., $G \in \{0, 1\}^{L-1}$ and its first component G_1 is the output of f at the root node.

The next step is to convert the vector G into a natural number that indicates the leaf node at which x arrives. In order to do this conversion, we propose the function

$$g : \bigcup_{N \in \mathbb{N}} \{0, 1\}^N \rightarrow \mathbb{N}$$

that associates a natural number to each boolean vector $y = (y_1, \dots, y_N)$ of any dimension $N \in \mathbb{N}$

$$g(y) = 2^N + \sum_{i=1}^N y_i 2^{N-i}.$$

This function g will convert a tree labeled in a boolean way into one labeled by natural numbers as it is shown in figure 3.15.

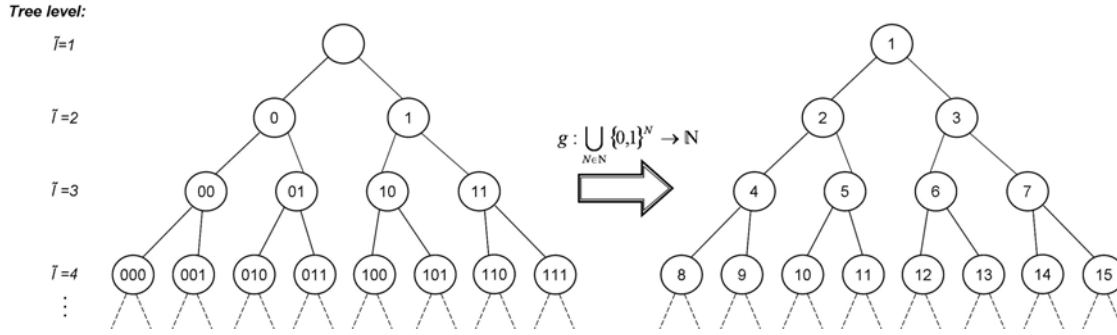


Figure 3.15: Left: outputs of the test at each node. Right: natural labeling of the nodes.

Remark: We consider $0 \in \mathbb{N}$ and by convention we take $\{0, 1\}^0 = \emptyset$ and $\sum_{i=1}^0 y_i 2^{N-i} = 0$. Thereby, we will have $g(\emptyset) = 1$, which means that the natural label of the root node is equal to 1, since at the root node we have no output of any test.

Remark: If we add a component $y_0 = 1$ to vector y then $\tilde{y} = (1, y)$ would be the input of a function $\tilde{g}: \bigcup_{N \in \mathbb{N}} \{0, 1\}^N \rightarrow \mathbb{N}$

$$g(y) := \tilde{g}(\tilde{y}) = \sum_{i=0}^N y_i 2^{N-i},$$

which is the equivalent to use $\{2^{n-i} \mathbf{e}_i : i = 1, \dots, N\}$ as basis, where $\{\mathbf{e}_1, \dots, \mathbf{e}_N\}$ is the standard basis of \mathbb{R}^N .

Path from the root to a leaf node At each level \tilde{l} of the DT T , the test corresponding to the node that x reaches has to be applied to x in order to know which node does x reach at the next level $\tilde{l} + 1$ of T . With this purpose, we propose a vectorial function F which indicates the path $n_1 P n_j$, $n_j \in V_{leaves}$, that x follows from the root to a leaf in a decision tree, i.e., the node x reaches at each level of T .

$$F: \mathcal{X} \rightarrow \bigcup_{N \in \mathbb{N}} \{0, 1\}^N$$

The components of F are defined recursively as follows:

$$F^1(x) = f_x(1)$$

$$F^i(x) = f_x\left(g((F^1, \dots, F^{i-1}))\right)$$

where $f_x(i) := f(x, i)$. Finally $F(x) = (F^1(x), \dots, F^{L-1}(x))$, being L the level of T , is a boolean vector that can be transformed into a natural number by applying the previously described function $g: \bigcup_{N \in \mathbb{N}} \{0, 1\}^N \rightarrow \mathbb{N}$. The output of g will indicate the leaf node n^* that x reaches. This leaf node is thus the result of the application of the composition of

the functions $g : \bigcup_{N \in \mathbb{N}} \{0, 1\}^N \longrightarrow \mathbb{N}$ and $F : \mathcal{X} \longrightarrow \bigcup_{N \in \mathbb{N}} \{0, 1\}^N$ previously described to x ,

$$n* = g(F(x)).$$

and the function $d_T : \mathcal{X} \longrightarrow \mathcal{C}$ is then

$$d_T(x) = l(g(F(x))),$$

where $l : V_{leaves} \longrightarrow \mathcal{C}$ ²⁰ is the terminal node class assignation function defined in section 3.13.

Usually, decision trees which represent the same classification function/rule $d_T : \mathcal{X} \longrightarrow \mathcal{C}$ (i.e., which assign the same class to a given measurement vector) are considered to be equal. However, the fact that two decision trees represent the same classification function does not imply that both decision trees have the same structure, i.e., that they are represented by the same graph $T = (V, E)$ and that the 2-tuples associated to each internal node (see definition 3.4.5) also coincide. For instance, figure 3.16 show an example of different structures of decision trees which represent the same classifier.

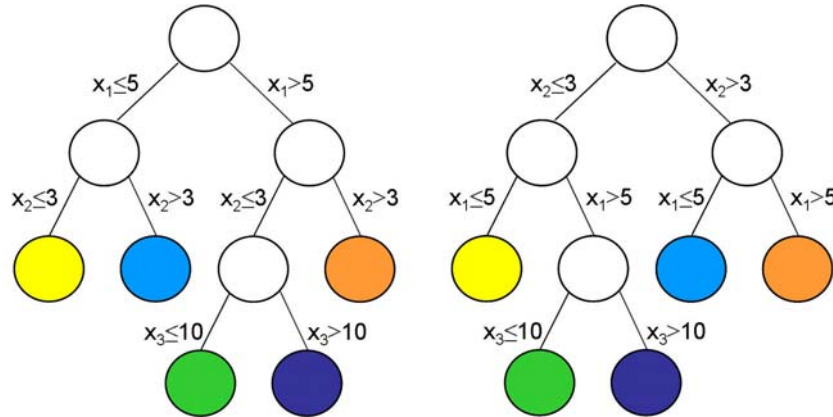


Figure 3.16: Example of two different decision trees which represent the same classifier

The structure of a decision tree is important to have an insight into the data. For instance, since decision trees have a hierarchical structure, the factors are prioritized and the most important features are supposed to be in the top levels of the trees. A metric to measure the difference between decision trees w.r.t. the structure becomes then necessary. We will see that such a metric is essential to study structural stability questions of decision trees.

²⁰The elements of $Nodes(T)$, and thereby the elements of V_{leaves} , are normally labeled with natural numbers. If not, we consider as domain of l the set $\{j \in \mathbb{N} \mid n_j \in V_{leaves}\}$

3.17 Distance between decision trees

In this section, we introduce a distance between two decision trees. This distance will measure the differences between the two trees, defined from the graph theory point of view (see chapter 2), as well as the differences between the associated 2-tuplas to each of their nodes (see chapter 3).

Notation We consider a decision tree as in the definition 3.4.2. The enumeration of the nodes will be the one we proposed in theorem 2.7.3, i.e.

$$E = \{(n_i, n_j) \in (V \setminus V_{leaves}) \times V \mid j = 2i \vee j = 2i + 1\}$$

(as an example, see figure 3.17). Let Ω be the space containing all possible decision trees defined in this way and \mathcal{D} the space of all possible learning samples.

Let $T_j = (V_j, E_j)$, $j = 1, 2$ be a decision tree where each node $i \in V_j$, $j = 1, 2$ has a 2-tuple $n_i^{T_j} = (q_i^{T_j}, \sigma_i^{T_j})$ associated. Let O_{T_j} be the order of the tree (see definition 2.2.1 in chapter 2).

Our aim in this chapter is to define a distance which measures the difference in the structure between two decision trees. With this purpose, we define this distance as the sum of the distances between the nodes of the trees, more concretely, between the 2-tuplas associated to each node. Thus, we are comparing the 2-tuples $n_i^{T_1} = (q_i^{T_1}, \sigma_i^{T_1})$ and $n_i^{T_2} = (q_i^{T_2}, \sigma_i^{T_2})$ associated to node $i \in V_j$, $j = 1, 2$. Since two trees do not necessarily have the same order, then we consider

$$i^* = \max\{i \mid n_i \in T_1 \vee n_i \in T_2\}$$

Then, we complete trees T_1 and T_2 such that $O(T_1) = O(T_2) = i^*$ (see figure 3.18) and assign a 2-tuple (I, M) to the nodes which are in T_1 but not in T_2 or vice versa. The values assigned to this 2-tuple are penalizing, in the sense that the distance between a node and this node with an assigned 2-tupla and will be equal to the maximal possible distance between two nodes.

Thus, without loss of generalization, we can suppose that the two trees have the same order.

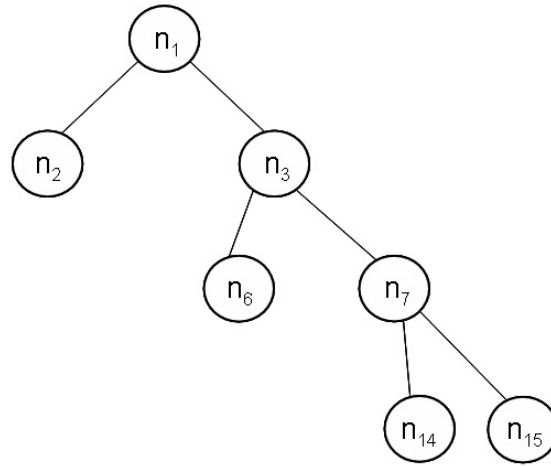
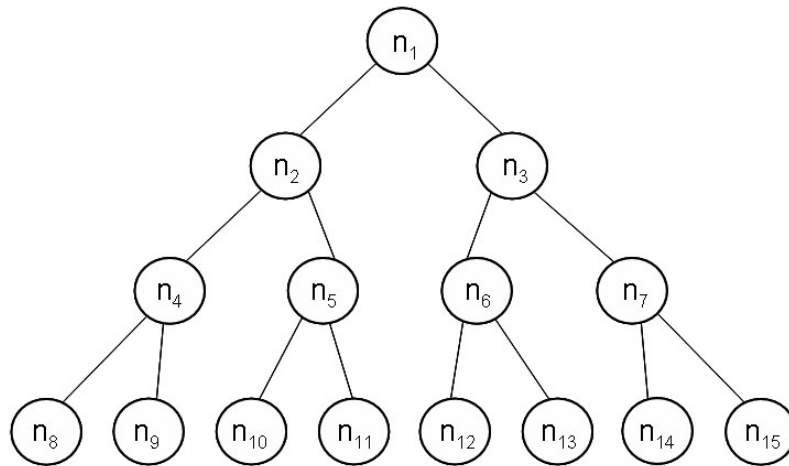
Definition 3.17.1: Let $T = (V, E)$ and $T' = (V', E')$ be two decision trees of the same order O where each node of T and T' has a 2-tuple $n_i = (q_i, \sigma_i) \in \Theta \times \mathbb{R}$ and $n'_i = (q'_i, \sigma'_i) \in \Theta \times \mathbb{R}$ associated, respectively. We define the n-distance between n_i and n'_i as a function

$$d_n : (\Theta \times \mathbb{R}) \times (\Theta \times \mathbb{R}) \rightarrow [0, +\infty)$$

$$d_n(n_i, n'_i) = d_n((q_i, \sigma_i), (q'_i, \sigma'_i)) := \sqrt{(d_v(q_i, q'_i))^2 + (d_s(n_i, n'_i))^2}$$

where $d_v : \Theta \times \Theta \rightarrow [0, +\infty)$ is defined as

$$d_v(q_i, q'_i) := \begin{cases} 0 & \text{if } q_i = q'_i \\ 1 & \text{else} \end{cases}$$

Figure 3.17: Example original DT T Figure 3.18: Complete tree T' corresponding to T

i.e., as the *discrete metric*, and $d_s : (\Theta \times \mathbb{R}) \times (\Theta \times \mathbb{R}) \rightarrow [0, +\infty)$ is defined as

$$d_s(n_i, n'_i) := \frac{|\sigma_i - \sigma'_i|}{1 + |\sigma_i - \sigma'_i|} \cdot (1 - d_v(q_i, q'_i)).$$

Lemma 3.17.1: Let $n = (q, \sigma)$, $n' = (q', \sigma') \in \Theta \times \mathbb{R}$. If we call

$$\tilde{d}_s(\sigma, \sigma') = \frac{|\sigma - \sigma'|}{1 + |\sigma - \sigma'|},$$

then $\tilde{d}_s : \mathbb{R} \times \mathbb{R} \rightarrow [0, 1]$ is a metric on \mathbb{R} . Moreover, for $d_s : (\Theta \times \mathbb{R}) \times (\Theta \times \mathbb{R}) \rightarrow [0, +\infty)$ it is fulfilled that

$$d_s(n, n') \in [0, 1]$$

and that if $q \neq q'$, then $d_v(q, q') = 1$ and $d_s(n, n') = 0$.

Proof. By definition of d_v , we have that

$$d_s((q, \sigma), (q', \sigma')) := \begin{cases} \frac{|\sigma - \sigma'|}{1 + |\sigma - \sigma'|} & \text{if } q = q' \\ 0 & \text{else.} \end{cases}$$

If we consider the euclidean distance $d_e(a, b) = |a - b|$ for all $a, b \in \mathbb{R}$, then \tilde{d}_s is a function of d_e . This function is a simple way to convert a given distance function into a bounded distance function. Let us see that \tilde{d}_s satisfies the requirements of a metric:

- a) $\tilde{d}_s(\sigma, \sigma') \geq 0$ $(= 0 \Leftrightarrow \sigma = \sigma')$ (Not negativity) (identity of indiscernibles)
- b) $\tilde{d}_s(\sigma, \sigma'') \leq \tilde{d}_s(\sigma, \sigma') + \tilde{d}_s(\sigma', \sigma'')$ (Triangular inequality)
- c) $\tilde{d}_s(\sigma, \sigma') = \tilde{d}_s(\sigma', \sigma)$ (Symmetry)

a) and c) hold trivially for \tilde{d}_s if they do for d_e : $\tilde{d}_s(\sigma, \sigma') = 0$ iff $|\sigma - \sigma'| = 0$. Also, $|\sigma - \sigma'| = |\sigma' - \sigma|$ implies $\tilde{d}_s(\sigma, \sigma') = \tilde{d}_s(\sigma', \sigma)$.

b) Note that the function $f(a) = a/(a + 1)$ is monotone increasing. From here, if, for example, $d_e(\sigma, \sigma'') \leq d_e(\sigma, \sigma')$ then also $\tilde{d}_s(\sigma, \sigma'') \leq \tilde{d}_s(\sigma, \sigma')$, in which case the triangular inequality is immediate. This is also true if $d_e(\sigma, \sigma'') \leq d_e(\sigma', \sigma'')$. Therefore, we can assume that $d_e(\sigma, \sigma'') > d_e(\sigma, \sigma')$ and $d_e(\sigma, \sigma'') > d_e(\sigma', \sigma'')$. Thus we obtain:

$$\begin{aligned} \tilde{d}_s(\sigma, \sigma'') &= \frac{d_e(\sigma, \sigma'')}{d_e(\sigma, \sigma'') + 1} \\ &\stackrel{d_e \text{ is a metric}}{\leq} \frac{d_e(\sigma, \sigma')}{d_e(\sigma, \sigma') + 1} + \frac{d_e(\sigma', \sigma'')}{d_e(\sigma', \sigma'') + 1} \\ &< \frac{d_e(\sigma, \sigma')}{d_e(\sigma, \sigma') + 1} + \frac{d_e(\sigma', \sigma'')}{d_e(\sigma', \sigma'') + 1} \\ &= \tilde{d}_s(\sigma, \sigma') + \tilde{d}_s(\sigma', \sigma'') \end{aligned}$$

Although d_e is not bounded whereas \tilde{d}_s is, when one is close to 0 so is the other. Thus both distances induce the same notion of nearness in the sense that sets closed in one are also closed in the other and vice versa. \square

Lemma 3.17.2: Let $a = (q, \sigma)$, $b = (q', \sigma')$, $c = (q'', \sigma'') \in \Theta \times \mathbb{R}$ such that $q = q'$ and $q \neq q''$ then:

$$\text{a) } 0 \leq d_n(a, b) \leq 1$$

$$\text{b) } d_n(a, c) = 1$$

and thus $d_n(a, b) \leq d_n(a, c)$, which in a decision tree context means that, if the splitting variables are different, then the distance is bigger than in the case they are the same and the threshold is different.

Proof. a) Let us see that $0 \leq d_n(a, b) \leq 1$. By definition 3.17.1, we have

$$\begin{aligned} d_n(a, b) &= \sqrt{(d_v(q, q'))^2 + (d_s(n, n'))^2} \\ &\stackrel{q=q'}{=} \sqrt{(d_s(n, n'))^2} \\ &= d_s(n, n') \in [0, 1] \quad \text{by lemma 3.17.1} \end{aligned}$$

b) It is trivial that $d_n(a, c) = 1$ since

$$\begin{aligned} d_n(a, c) &= \sqrt{(d_v(q, q''))^2 + (d_s(n, n''))^2} \\ &= \sqrt{(d_v(q, q''))^2} = 1 \end{aligned}$$

\square

Proposition 3.17.1: Let $d_n : (\Theta \times \mathbb{R}) \times (\Theta \times \mathbb{R}) \rightarrow [0, +\infty)$ be the function depicted in definition 3.17.1 and $\Theta = \{1, 2, \dots, h\}$, where h is the number of variables of the measurement space, then d_n is a metric on $\Theta \times \mathbb{R}$. Thus, $(\Theta \times \mathbb{R}, d_n)$ is a metric space.

Proof. Let us see that d_n satisfies the requirements of a metric:

- a) $d_n(n, n') \geq 0$ $(= 0 \Leftrightarrow n = n')$ (Not negativity) (identity of indiscernibles)
- b) $d_n(n, n'') \leq d_n(n, n') + d_n(n', n'')$ (Triangular inequality)
- c) $d_n(n, n') = d_n(n', n)$ (Symmetry)

a) and c) are trivially fulfilled.

b) Let us see that $d_n(\overbrace{(q, \sigma)}^n, \overbrace{(q', \sigma')}^{n'}) + d_n(\overbrace{(q', \sigma')}^{n'}, \overbrace{(q'', \sigma'')}^{n''}) \geq d_n(\overbrace{(q, \sigma)}^n, \overbrace{(q'', \sigma'')}^{n''})$

Case 1: $q = q'$ then, by lemma 3.17.2, $d_n(n, n') \in [0, 1]$

Case 1.1: $q = q' = q'' \Rightarrow d_n(n, n') = \tilde{d}_s(\sigma, \sigma'), d_n(n', n'') = \tilde{d}_s(\sigma', \sigma'')$
and $d_n(n, n'') = \tilde{d}_s(\sigma, \sigma'')$.

Since \tilde{d}_s is a metric on \mathbb{R} (see lemma 3.17.1), it fulfills the triangular inequality.

Case 1.2: $q = q' \wedge q \neq q'' \Rightarrow d_n(n, n') + d_n(n', n'') > 1 = d_n(n, n'')$
since $d_n(n, n') \in [0, 1]$ and $d_n(n', n'') = 1$

Case 2: $q \neq q'$ then, by lemma 3.17.2, $d_n(n, n') = 1$

Case 2.1: $q = q'' \wedge q' \neq q'' \Rightarrow d_n(n, n') + d_n(n', n'') = 1 + 1 > d_n(n, n'')$
 $d_n(n, n'') \in [0, 1]$

Case 2.2: $q \neq q'' \wedge q' \neq q'' \Rightarrow d_n(n, n') + d_n(n', n'') = 1 + 1 > d_n(n, n'') = 1$

Case 2.3: $q \neq q'' \wedge q' = q'' \Rightarrow$
 $d_n(n, n') + d_n(n', n'') = 1 + d_n(n', n'') > d_n(n, n'') = 1$
since by lemma 3.17.2 $d_n(n', n'') \in [0, 1]$

□

Definition 3.17.2: Let us suppose two decision trees $T = (V, E)$, $T' = (V', E')$ of the same order O where each node of T and T' has a 2-tuple $n_i = (q_i, \sigma_i) \in \Theta \times \mathbb{R}$ and $n'_i = (q'_i, \sigma'_i) \in \Theta \times \mathbb{R}$ associated, respectively. We define the **T-distance** between T and T' as a function

$$\delta : \Omega \times \Omega \rightarrow \mathbb{R}^+$$

$$\delta(T, T') := \sum_{i=1}^O d_n(n_i, n'_i) := \sum_{i=1}^O (d_n((q_i, \sigma_i), (q'_i, \sigma'_i)))$$

where d_n is the distance between nodes of a decision tree described in definition 3.17.1.

Proposition 3.17.2: Let us suppose that the same conditions as in the previous definition are fulfilled, then δ is a metric on Ω . Thus, (Ω, δ) is a metric space.

Proof. We prove that δ satisfies the requirements of a metric:

a) $\delta(T, T') \geq 0$ ($= 0 \Leftrightarrow n = n'$) (Not negativity) (identity of indiscernibles)

$$\delta(T, T') = \sum_{i=1}^O d_n(n_i, n'_i) \geq 0$$

since $\text{range}(d_n) = [0, 1]$.

b) $\delta(T, T'') \leq \delta(T, T') + \delta(T', T'')$ (Triangular inequality)

$$\begin{aligned} \delta(T, T'') &= \sum_{i=1}^O d_n(n_i, n''_i) \\ &\stackrel{\substack{d_n \text{ is a metric} \\ d_n \geq 0}}{\leq} \sum_{i=1}^O (d_n(n_i, n'_i) + d_n(n'_i, n''_i)) \\ &= \sum_{i=1}^O d_n(n_i, n'_i) + \sum_{i=1}^O d_n(n'_i, n''_i) \end{aligned}$$

since d_n satisfies the triangular inequality.

c) $\delta(T, T') = \delta(T', T)$ (Symmetry)

$$\delta(T, T') = \sum_{i=1}^O d_n(n_i, n'_i) = \sum_{i=1}^O d_n(n'_i, n_i) = \delta(T', T)$$

□

3.18 Instability coefficient

The instability problem with tree-structured classifiers is that they may be sensitive to small changes in the training sample. That is, tree classification algorithms may produce dramatically different decision trees if the training sample is slightly changed. The instability problem complicates the process of the knowledge discovery because the users are disturbed by the different decision trees generated from almost the same input learning samples.

The structural instability refers to the changes produced in the structure of the decision tree, i.e., in the tree $T = (V, E)$ from the graph theory point of view and in the split variable and threshold associated to each internal node (see definition 3.4.5).

Our approach One possible conceptional approach to deal with stability questions is to derive an instability coefficient. For this, we need two metric spaces (Ω, δ) and (\mathcal{D}, d_D) (see section 3.17), where δ and d_D denotes the metric on the sets Ω and \mathcal{D} , respectively. Let $\Xi : \mathcal{D} \rightarrow \Omega$ be the function that assigns a DT $T \in \Omega$ to each learning sample $D \in \mathcal{D}$. This DT is the result of a certain supervised learning method for the construction of tree structured classifiers and we define the **instability coefficient** α of this supervised learning method as

$$\alpha = \min\{\alpha' \in \mathbb{R}^+ | \delta(\Xi(D), \Xi(D')) \leq \alpha' d_D(D, D')\}$$

i.e.

$$\alpha = \min\{\alpha' \in \mathbb{R}^+ | \delta(T, T') \leq \alpha' d_D(D, D')\}.$$

If the function $\Xi : \mathcal{D} \rightarrow \Omega$ is Lipschitz continuous, i.e., if there exists a constant $K \geq 0$ such that for all $D, D' \in \mathcal{D}$

$$\delta(\Xi(D), \Xi(D')) \leq K d_D(D, D')$$

then, **instability coefficient** of the supervised learning method correspond to the **Lip-schitz constant** of the function Ξ .

There are several distances between databases defined in the literature, as a general example see [45]. Most of them are based on the Fréchet distance (see [11]), which is one of the most fundamental measures used to compute dissimilarity between two sequences of points. Other possible distances used to measure the proximity of databases are Hausdorff distance (see [56]), Eath-Mover distance (see [57]), etc. Let us briefly see an example of one of them:

Hausdorff distance: Named after Felix Hausdorff (1868-1942), Hausdorff distance is the “maximum distance of a set to the nearest point in the other set”. More formally, Hausdorff distance from set A to set B is a maximin function, defined as

$$H(A, B) = \max_{a \in A} \{ \min_{b \in B} (d(a, b)) \}$$

where a and b are points of sets A and B respectively, and $d(a, b)$ is any metric between these points.

Note: As we have already explained along this chapter, decision trees make a partition of the measurement space (see figure 3.19 as summary). From now on, we will eventually call them **crisp** decision trees, in order to distinguish these decision trees from the ones we introduce in the next chapter, where the partition of the space induced by the tree is a fuzzy partition.

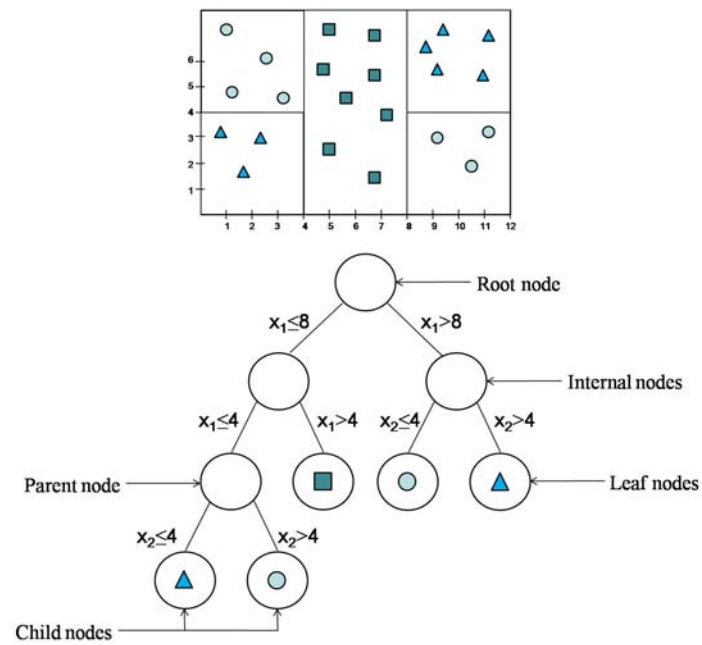


Figure 3.19: Decision Tree example

Chapter 4

SODT (Soft Operators Decision Tree)

4.1 Introduction

In this chapter, we introduce a model (SODT) which, due to the best of our knowledge, has not been considered before in the literature. SODT integrates decision trees with “soft operators”, mainly with the soft comparison operators proposed by Mlynski in his work [43]. The aim of this combination is to generate classifiers from training data through a process of recursively “soft” splitting of the data space. Recall that tree classifiers partition the measurement space in a data driven manner to finally assign a class to each of the non-overlapping subspaces in the partition.

“Soft splitting” in our case refers to a partition of the space where the operators \leq, \geq are not necessarily Boolean (which creates a sharp/crisp division), but somehow fuzzified. Thus, the elements around the boundaries of the partition sets belong to these different sets with certain degrees of membership instead of purely belong to one of them, i.e., the sets of the partition overlap (see figure 4.1).

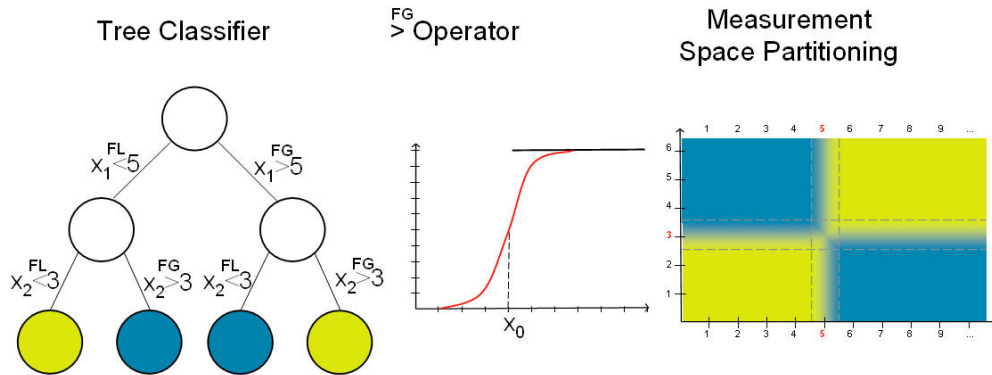


Figure 4.1: example of the soft binary partitioning of the space made by a SODT

Tree classifiers that combine fuzzy theory and decision trees, creating thus this partition of the space into overlapping regions, are commonly known as fuzzy decision trees. There

exist several examples in the literature ([2], [3], [68], [7], [35], [33], [1], [47]) that propose combinations of this type. The intent is always to exploit complementary advantages of both: popularity in applications to learning from examples and high knowledge comprehensibility of decision trees combined with the ability to deal with inexact and uncertain information of fuzzy representation. However, one can deal with the problem in many different ways.

Concerning the creation of the classifier, the main difference to other fuzzified decision trees is that in our case, the fuzzification is not done in the nodes, with a division of the domain of definition of each variable in fuzzy sets, but it is done in the operators \leq , \geq , avoiding in this way the division of the domain space into fuzzy sets and hence saving parameters. Another remarkable difference is that we use the Gini index (see section 3.7), adapted to the soft operators, to decide where to split at each node.

A further aspect of this inclusion of soft operators is that we choose them in a way so that the resulting goodness function (used by this method) is differentiable and thus allows to calculate the best split points by means of gradient descent methods, as well as to optimize fuzzy parameters and split points simultaneously.

4.2 Outline of the chapter and background

In the current chapter we are basically concerned with the presentation of our classification model (SODT), its construction, application and the consistency of the mathematical formulation behind this. One of the main points of this chapter, apart from the definition of a SODT itself in section 4.3, is the notion of degree of membership of an element of the measurement space to each of the nodes of a SODT. A definition of this notion is given in section 4.4 and its consistency and asymptotical behaviour are analyzed. In section 4.5 the learning and inference process are depicted and summarized in figure 4.9. Next sections 4.6 - 4.9 expound in detail the learning process. Among them, the SODT induction section is of special interest since it involves the creation of SODT node impurity functions, which are the basics for the splitting procedure of our tree classifier. To finalize the learning process, its algorithm is outlined in section 4.10. In order to classify new measurement vectors with our model, the SODT inference and its two main options are described in section 4.11. And finally, in order to compute the accuracy of our model, the SODT misclassification rate is explained in section 4.12.

In the special case that the data sample is given with uncertainty (fuzzified data sample), the variant of SODT explained in section 4.13 deals with this type of data.

Finally, in section 4.14, we propose a distance to measure dissimilarities between two SODTs.

Types of variables and types of trees

ID3 and CART are the two most important discriminative learning algorithms working by recursive partitioning. Their basic ideas are the same: partition the sample space in a

data-driven manner, and represent the partition as a tree.

Nevertheless, the two trees are different. ID3 assumes discrete domains with small cardinalities. This is an advantage as it increases comprehensibility of the induced knowledge, but may require an a priori partitioning. The CART algorithm does not require prior partitioning. The conditions in the tree are based on thresholds for continuous domains (see definition 3.3.3), which are dynamically computed.

The type of measurement vectors we are going to consider are composed of continuous ordered variables and belong to a measurement space $\mathcal{X} \subseteq \mathbb{R}^h$. For this reason we choose the methodology from CART as the basis of the decision trees of our model, which integrates decision trees and soft operators. In chapter 3 we explained in detail the model of (crisp) decision tree (based on CART) on which we are based in SODT.

The soft operators are adapted in order to avoid the sharp division of the measurement space produced by (crisp) decision trees and to allow the implementation of linguistic rules like for example: given an element $o \in \mathcal{X}$, the greater the attribute o_i , the more sure we are that this element belongs to class c_k .

Fuzzy decision trees

In the past, several variants of fuzzy decision trees were introduced by different authors ([2], [3], [68], [7], [35], [33], [1], [47]).

We observe that they can be distinguished according to the type of learning data considered, the data preprocessing phase, and the method to select the split points. For instance, [47] consider fuzzy data, i.e., data with two possible output classes where the outputs are given in form of degree of membership to one of the classes, as learning sample for their model and do not require a preprocessing of the data whereas [68] consider crisp data and fuzzify them in a preprocessing phase. Models that consider fuzzy data normally deal with them in a similar way as regression trees, where a label that approximates the output is assigned to each node and formulas for computing the error are applied. The parameters (split points and additional fuzzy parameters) chosen at each node will be the ones that minimize these formulas.

The fact that the fuzzification in our model is done in the operators instead of in the variables has the advantage that the data do not need a preprocessing phase, even if the output of the learning data is given in crisp form.

Concerning the method to select the split points, [33] adapts the C4.5 algorithm, while [1], [2] and [35] for example, adapt the well known ID3 algorithm so that it works with fuzzy sets. As we have already explained in the previous section, the ID3 algorithm assumes discrete domains, which are normally given in form of linguistic variables, thus, the models that combine ID3 with fuzzy theory transform the linguistic variables into fuzzy sets.

Notation and definitions

For the learning sample \mathcal{L} and its elements we will take the same notations that we proposed in chapter 3 section 3.3.

In the following of the thesis, we restrict ourselves to a finite learning sample $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times \mathcal{C}$,¹ with continuous variables/features (see section 3.3.1), i.e., variables that can assume any value from an interval of the real numbers. It means that the measurement space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_h$, where each \mathcal{X}_i , $i = 1, \dots, h$, is of the form:

$$[a, b], \quad [a, b), \quad (a, b], \quad (a, b), \quad (a, +\infty), \quad [a, +\infty), \quad (-\infty, b), \quad (-\infty, b]$$

for $a, b \in \mathbb{R}$.

Our aim is to construct a stable, accurate and interpretable classifier which represents the concept² underlying the learning sample \mathcal{L} and which can deal with uncertainty.

With that purpose, we propose SODT, which is a new model that integrates decision trees (which confer interpretability and accuracy to our model) with soft comparison operators (which confer stability and the ability to deal with uncertainty).

We will explain in the next sections that the SODT can also be considered as a “fuzzy classifier”. Let us introduce this conception.

Recall that a classification problem can be defined as a pair $(\mathcal{X}, \mathcal{C})$, where \mathcal{X} , called the measurement space, is the collection of all possible measurements, and $\mathcal{C} = \{1, 2, \dots, J\}$ is the set of all possible classes (see section 3.3). A classifier maps each measurement vector into exactly one class whereas a fuzzy classifier maps it into a J-dimensional vector with possibilities that the given measurement vector belongs to the corresponding classes. Formally,

Definition 4.2.1: Let $(\mathcal{X}, \mathcal{C})$ be a classification problem. A **Fuzzy classifier** is a function $\Psi(\cdot)$ defined on \mathcal{X} so that for every $x \in \mathcal{X}$, $\Psi(x)$ is equal to a vector (p_1, \dots, p_J) , where p_i represents the possibility that a given instance x belongs to class $i \in \mathcal{C} = \{1, \dots, J\}$. For ease of representation, the function

$$\Psi : \mathcal{X} \longrightarrow \{(p_1, \dots, p_J) \mid p_i \in [0, 1]\}.$$

is sometimes represented as a vector of functions

$$(\psi_1, \dots, \psi_J)$$

where ψ_i is a possibility function $\mathcal{X} \longrightarrow [0, 1]$. For any given instance $x \in \mathcal{X}$, the relation $\psi_i(x) > \psi_j(x)$ indicates that it is more likely for the instance x to be in class i than in class j .

4.3 Definition SODT

Analogously to the formal definition of a decision tree (definition 3.4.5) we gave in section 3.4.2, we will define a SODT, which is also a binary-tree structured classifier where

¹ $X \in \mathcal{M}_{N \times h}$, $Y \in \mathcal{M}_{N \times 1}$, being N the number of elements of the learning sample. It can also be written as $\mathcal{L} = \{(x^{(k)}, y^{(k)}) : k \in \{1, \dots, N\}\}$

²see section 3.3

the components are basically the same as in a crisp DT, but here an "unpreciseness" factor m is added to model the $>$ and $<$ soft comparison operators. First of all, let us define the soft comparison operators.

Definition 4.3.1: The soft comparison operators ([43], [42]), which are called *Fuzzy Less* (FL) and *Fuzzy Greater* (FG), are defined as

$$FL : \mathbb{R}^2 \times (0, +\infty) \longrightarrow [0, 1]$$

$$FL(x, y, m) := \frac{\tanh((y - x) \cdot m) + 1}{2}$$

$$FG : \mathbb{R}^2 \times (0, +\infty) \longrightarrow [0, 1]$$

$$FG(x, y, m) := \frac{\tanh((x - y) \cdot m) + 1}{2}$$

where m is used to model the "unpreciseness" of the operators.³

Lemma 4.3.1: Let $(o, \sigma, m) \in \mathbb{R}^2 \times (0, +\infty)$ and let $FL : \mathbb{R}^2 \times (0, +\infty) \longrightarrow [0, 1]$ and $FG : \mathbb{R}^2 \times (0, +\infty) \longrightarrow [0, 1]$ be the Fuzzy less and Fuzzy greater operators presented in definition 4.3.1. Then

$$FL(o, \sigma, m) + FG(o, \sigma, m) = 1 \quad (4.1)$$

Proof.

$$\begin{aligned} FL(o, \sigma, m) + FG(o, \sigma, m) &= \frac{\tanh((\sigma - o) \cdot m) + 1}{2} + \frac{\tanh((o - \sigma) \cdot m) + 1}{2} \\ &= \frac{\frac{e^{(\sigma-o) \cdot m} - e^{-(\sigma-o) \cdot m}}{e^{(\sigma-o) \cdot m} + e^{-(\sigma-o) \cdot m}} + 1}{2} + \frac{\frac{e^{-(\sigma-o) \cdot m} - e^{(\sigma-o) \cdot m}}{e^{-(\sigma-o) \cdot m} + e^{(\sigma-o) \cdot m}} + 1}{2} \\ &= 1 \end{aligned}$$

□

As an illustration of this lemma, see figure 4.2.

Definition 4.3.2: Let $T = (V, E)$ be a rooted ordered binary tree, and let $V = V_1 \dot{\cup} \dots \dot{\cup} V_L$ be a partition where each $V_i, i = 1, \dots, L$ contains the vertices or nodes at each *level* of the tree (see theorem 2.7.2), for instance the elements of V_1 are the sources of the tree. A SODT is composed of the tree T and a family of 3-tuples (q_i, σ_i, m_i) , each associated to one of the non-leaves vertices $n_i \in V \setminus V_{\text{leaves}}$.

For each node $n_i \in V \setminus V_{\text{leaves}}$ we have:

³In fact, any sigmoid function can be used as basis for the soft comparison operators.

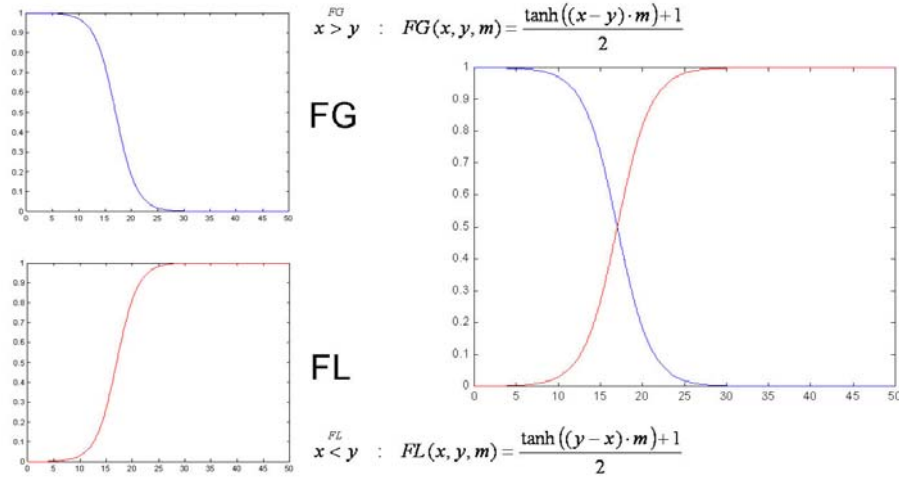


Figure 4.2: FL and FG operators sum up to one

- $q_i \in \{1, \dots, h\}$, where $h \in \{1, 2, \dots\}$
- $\sigma_i \in \mathcal{X}_{q_i} \subseteq \mathbb{R}$
- $m_i \geq 0$

Further, there exists a map

$$l : V_{leaves} \longrightarrow \mathcal{C}$$

which assigns a class label $l(n_i) \in \mathcal{C}$ to each terminal node $n_i \in V_{leaves}$. In other words, each $n_i \in V_{leaves}$ has an element $l(n_i) \in \mathcal{C}$ associated ⁴.

Remark: A SODT is a classifier that works by means of recursive “soft splitting/ partition” a measurement space $\mathcal{X} \subseteq \mathbb{R}^h$ to finally assign a class label to each subspace in the partition. New instances will be classified according to the subspace to which they belong. “Soft partition” refers to a partition of the space into subsets with overlapping regions, i.e., an instance may belong to several subspaces with certain degrees of membership. At each node, a “soft bipartition” of the subspace is done in a similar way as we explained in 3.4.1 for the crisp case but instead of the usual $<$ and $>$ operators, “soft comparison operators” are used (see definition 4.3.1). These operators will be responsible for the soft partition. The components of the 3-tuple associated to each internal and root node in a SODT have the following meaning:

- $q_i \in \{1, \dots, h\}$ represents a *feature axis*, i.e., x_{q_i} will be the variable used for splitting in node n_i (*splitting variable* at node n_i),
- $\sigma_i \in \mathcal{X}_{q_i}$ is the *threshold value* of the variable x_{q_i} , i.e., the point of the domain of definition of the variable x_{q_i} where the space is divided,

⁴Function $l : V_{leaves} \longrightarrow \mathcal{C}$ is explained in detail in section 4.9

- $m_i \geq 0$ is a parameter which will be used to model the *unpreciseness* of the soft comparison operators (see definition 4.3.1).

We call this parameter m_i *unpreciseness factor* and the 3-tuple (q_i, σ_i, m_i) *split point* at node n_i .

Graphical representation of a SODT

Figure 4.3 illustrates definition 4.3.2. In this figure, $T = (V, E)$ where $V = \{n_1, n_2, n_3, n_4, n_5\}$.

$V_{leaves} = \{n_3, n_4, n_5\}$ and the elements of $V \setminus V_{leaves} = \{n_1, n_2\}$ have the 3-tuples $(2, \sigma_1, m_1)$ and $(1, \sigma_2, m_2)$ associated to n_1 and n_2 respectively. This 3-tuples represent the partition of the measurement space shown in figure 4.5. The *fuzzy less* and *fuzzy greater* operators are illustrated in figure 4.4.

For this illustrative example, we consider the set of classes $\mathcal{C} = \{c_1, c_2, c_3\}$. The class labels assigned to the leaf nodes are $l(n_3) = c_1$, $l(n_4) = c_2$ and $l(n_5) = c_3$.

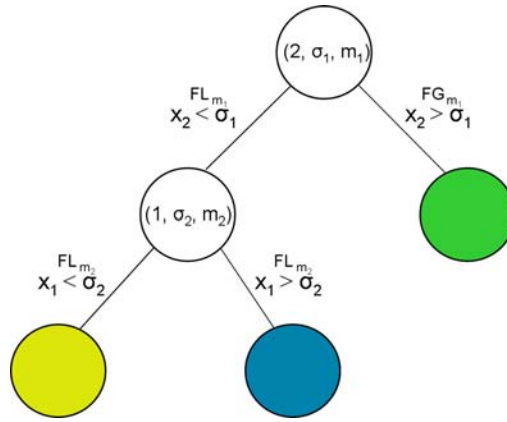


Figure 4.3: Example of a SODT

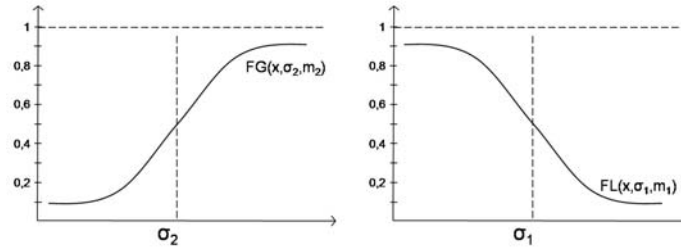


Figure 4.4: Fuzzy Greater and Fuzzy Less operators of the previous example of a SODT

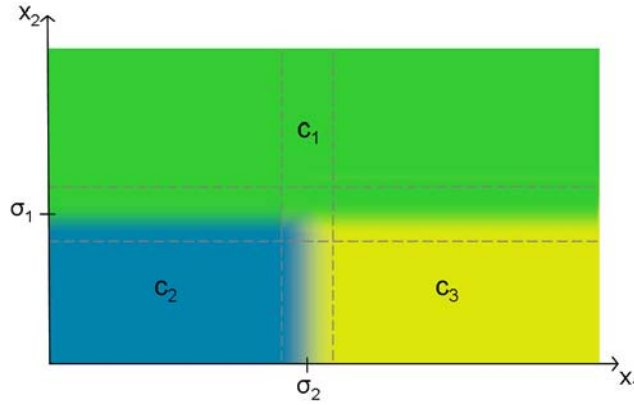


Figure 4.5: Space partitioning of the previous example of a SODT

4.4 Membership degree of an element $o \in \mathcal{X}$ to each of the SODT nodes

Introduction

To each node n of a (crisp) decision tree we can associate a subset R_n (see section 3.4.2) of the measurement space \mathcal{X} . The set R_n itself is a subset of the set $R_{P(n)}$ associated to the parent node $P(n)$ of n , and it comprises those elements of $R_{P(n)}$ which fulfill the condition imposed by the split point at the parent node. We thus have the characteristic function

$$f_n : \mathcal{X} \longrightarrow \{0, 1\}$$

associated to this node, where an element $x \in \mathcal{X}$ belongs to the set R_n if and only if $f_n(x) = 1$ and does not belong to R_n if and only if $f_n(x) = 0$.

A concept introduced by Lotfy Zadeh in 1965 is the *degree of membership* of an element to a *fuzzy set* (see appendix A).

In our SODT, every element $x \in \mathcal{X}$ belongs to a node n with a certain *degree of membership* and the function associated to this node could be

$$\hat{f}_n : \mathcal{X} \longrightarrow [0, 1]$$

This fuzzification of our model is not done using membership functions for the domain \mathcal{X} of \hat{f}_n , i.e., it is not done dividing \mathcal{X} into different fuzzy sets. Instead it is done applying membership functions for the operators responsible for the split of each node.

Under each internal node, a test appears as a condition on a single attribute at a time, regarding a parameter m , which determines the fuzzification of the \leq and \geq operators. This parameter m will set the difference between crisp DT and SODT.

Definition 4.4.1: Given a SODT denoted by $T = (V, E)$, let $o \in \mathcal{X}$ be the measurement vector of an object (o, c) of the learning sample $\mathcal{L} \subseteq \mathcal{X} \times \mathcal{C}$ and let $n_i \in V$ be a node of the

SODT T . We define the *degree of membership* (MSD) $\mu_{n_i}(o)$ of the measurement vector $o \in \mathcal{X}$ to the node $n_i \in V$ in a recursive manner via the function

$$g : \mathcal{X} \times I_V \times V \times [0, 1] \longrightarrow [0, 1]$$

where $I_V := \{k \in \mathbb{N} \mid n_k \in V\}$ and so, the initial value $\mu_{n_1}(o)$ is taken to be 1 and for all $k \in I_V \setminus \{1\}$, $\mu_{n_k}(o)$ is taken as follows

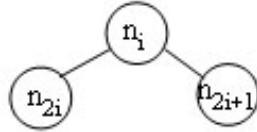
$$\begin{aligned} \mu_k(o) &:= \mu_{n_k}(o) := g(o, k, n_{P_k}, \mu_{P_k}(o)) \\ &= \begin{cases} FL(o_{q_{P_k}}, \sigma_{P_k}, m_{P_k}) \cdot \mu_{P_k}(o) & \text{if } k \equiv 0 \pmod{2} \\ FG(o_{q_{P_k}}, \sigma_{P_k}, m_{P_k}) \cdot \mu_{P_k}(o) & \text{if } k \equiv 1 \pmod{2} \end{cases} \end{aligned}$$

where $FL : \mathbb{R}^2 \times (0, +\infty) \longrightarrow [0, 1]$ and $FG : \mathbb{R}^2 \times (0, +\infty) \longrightarrow [0, 1]$ are the *Fuzzy Less* and *Fuzzy Greater* operators described in definition 4.3.1.

For each $n_k \in V \setminus V_1$ we define $P_k := \{k \in I_V \mid n_{P_k} \in P(n_k)\}$ and, by definition of $P(i)$ in chapter 2, n_{P_k} exists, is unique and an element of V . Thus, $P_k \neq \emptyset$ is a set with exactly one element, so from now on we will identify P_k with its unique element. In other words, P_k is the label of the direct predecessor or parent of n_k .

Remark: By labeling of a binary tree as in chapter 2 (see theorem 2.7.3), the label P_k of the parent of n_k is:

$$P(n_k) \ni n_{P_k} = \begin{cases} n_{\frac{k}{2}} & \text{if } k \equiv 0 \pmod{2} \\ n_{\frac{k-1}{2}} & \text{if } k \equiv 1 \pmod{2}. \end{cases} \quad (4.2)$$



Parent-children labels in a SODT. $P(n_{2i}) = P(n_{2i+1}) = n_i$ and $P_{2i} = P_{2i+1} = i$

Remark: The *degree of membership* of any object $o \in \mathcal{X}$ to the root node $n_1 \in V$ is assumed to be one, $\mu_{n_1}(o) = 1$. The reason of this assumption is that, since at the root node no split or division of the measurement space \mathcal{X} is done, we assume that the elements of $o \in \mathcal{X}$ belong completely to this node.

Graphical representation of the Membership degree (MSD): In the following figures 4.6 and 4.7 we show the degree of membership of the one-dimensional measurement vectors of the set $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ to the right and left node if we split with a threshold value $\sigma = 1.5$. In figure 4.6 we consider a crisp decision tree, i.e., the elements either do or do not belong to a node. If an element belongs to a node, the degree of membership of this element to the node is considered to be 1 and if not, it is considered to be 0. In figure 4.7,

a SODT is considered, with unpreciseness factor $m = 0.8$. In both cases, we can observe that the sum of the degrees of membership to the right and to the left child nodes is equal to one (see proposition 4.4.4).

In these figures we represent the division of the root node, where the degree of membership of every element to the root node is taken as 1 by assumption.

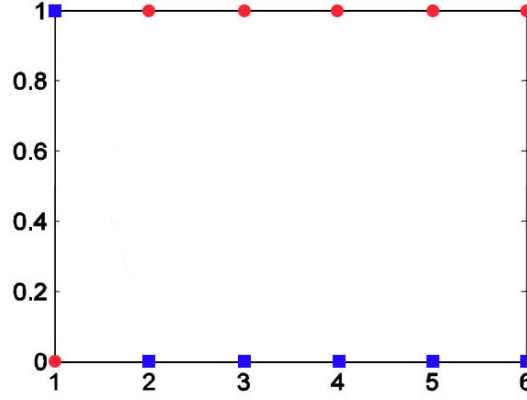


Figure 4.6: Example MSD of the elements of $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ to nodes n_L (blue squares) and n_R (red circles) in CRISP DT for a threshold value $\sigma = 1.5$.

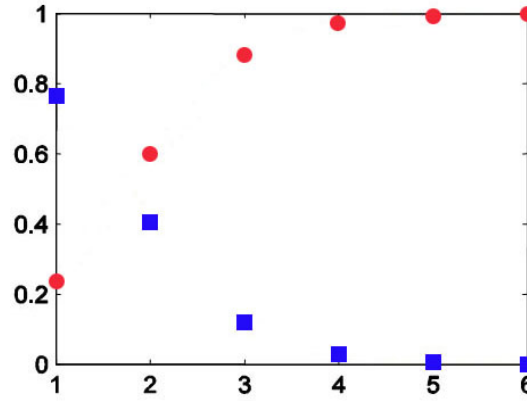


Figure 4.7: Example MSD of the elements of $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ to nodes n_L (blue squares) and n_R (red circles) in SODT for a threshold value $\sigma = 1.5$ and an unpreciseness factor $m = 0.8$.

Notation: From now on, we will substitute $\mu_{n_i}(o)$ by $\mu_i(o)$ for simplicity of notation.

In order to emphasize the unpreciseness factor m , we will write sometimes $\mu_i(o, m)$ instead of $\mu_i(o)$, being m the unpreciseness factor associated to the parent node of n_i , i.e., the third component of the 3-tuple associated to $P(n_i)$.

In terms of left and right child nodes of a node $n \in \text{Nodes}(T)$, with 3-tuple (o_{q_n}, σ_n, m_n) associated, the definition of the degree of membership of an object $o \in \mathcal{X}$ to the left n_L and right n_R child nodes of the node n , knowing the degree of membership of this element o to the parent node n ($\mu_n(o)$), would be:

$$\mu_{n_L}(o) = FL(o_{q_n}, \sigma_n, m_n) \mu_n(o)$$

and

$$\mu_{n_R}(o) = FG(o_{q_n}, \sigma_n, m_n) \mu_n(o),$$

respectively.

Well defined range of the degree of membership function

In order to prove the consistency of definition 4.4.1 of the membership degree of an element $o \in \mathcal{X}$ to each node $n \in V$ of a SODT $T = (V, E)$, we suggest the following proposition. This proposition states that the range of the function $g : \mathcal{X} \times I_V \times V \times [0, 1] \longrightarrow [0, 1]$ in this definition is indeed well-defined, i.e., that the image of g is really contained in $[0, 1]$.

Proposition 4.4.1: Let $T = (V, E)$ be a SODT and \mathcal{X} a measurement space. Then, for every $n_k \in V$ and every $o \in \mathcal{X}$, it is fulfilled that $\mu_k(o) \in [0, 1]$.

Proof. By induction.

For $k = 1$, $\mu_1(o) = 1 \in [0, 1]$ for all $o \in \mathcal{X}$ by definition.

If P_k fulfills the proposition, $\mu_{P_k} \in [0, 1]$, then by the recursive definition of μ_k

$$\mu_k(o) := \mu_{n_k}(o) := g(o, k, n_{P_k}, \mu_{P_k}(o))$$

we can distinguish between two cases:

$$\begin{aligned} k \equiv 0 \pmod{2} \implies \quad & \mu_k(o) = \mu_{n_k}(o) = FL(o_{q_{P_k}}, \sigma_{P_k}, m_{P_k}) \cdot \mu_{P_k}(o) \\ & \text{which belongs to } [0, 1] \text{ because } \text{range}(FL) = [0, 1] \text{ and} \\ & \mu_{P_k} \in [0, 1] \text{ by hypothesis of induction.} \end{aligned}$$

$$\begin{aligned} k \equiv 1 \pmod{2} \implies \quad & \mu_k(o) = \mu_{n_k}(o) = FG(o_{q_{P_k}}, \sigma_{P_k}, m_{P_k}) \cdot \mu_{P_k}(o) \\ & \text{and analogously it is proved that } \mu_k(o) \in [0, 1] \\ & \text{because } \text{range}(FG) = [0, 1] \end{aligned}$$

□

4.4.1 Measures of fuzzy cardinality

The aim of this section is to define a measure of the measurement vectors set $X \subseteq \mathcal{X}$ at each node that is the analogue to the counting measure in the crisp case, which indicates the number of objects of the learning sample at a node of the (crisp) decision tree. We will distinguish between two cases: X finite and X Borel subset of \mathcal{X} . The first case is for every given learning sample $\mathcal{L} = (X, Y)$, which are always finite. The second case allows the incorporation of a priori information given by experts, like for example “ If *attribute* q belongs to the interval $[a, b]$, then this object belongs to class c ”. In this example, $[a, b]$ is a Borel set but it is not finite.

P_n measure on σ -algebra \mathcal{F}_X

Lemma 4.4.1: Let X be a finite subset of our measurement space $\mathcal{X} \subseteq \mathbb{R}^h$ and let us call \mathcal{F}_X the power set of X (see appendix B), then (X, \mathcal{F}_X) is a **measurable space**, i.e., \mathcal{F}_X is a σ -algebra on X . Moreover, \mathcal{F}_X is the largest σ -algebra on X .

Proof. It is trivial that \mathcal{F}_X fulfills the conditions of the definition of a σ -algebra (or σ -field).

1. $\emptyset \in \mathcal{F}_X$.
2. If $A \in \mathcal{F}_X$, then the complement $A^c \in \mathcal{F}_X$.
3. If $A_j \in \mathcal{F}_X$, $j = 1, 2, \dots$, then their union $\bigcup A_j \in \mathcal{F}_X$

□

Thus we can define a measure on X .

Definition 4.4.2: Let T be a SODT and let (X, \mathcal{F}_X) be the measurable space defined above. For each node $n \in V(T)$ we define a set function on \mathcal{F}_X , $P_n : \mathcal{F}_X \longrightarrow \mathbb{R}$, as $P_n(A) = \sum_{x \in A} \mu_n(x)$ for all $A \subseteq X$, where $\mu_n : \mathcal{X} \longrightarrow [0, 1]$ is given for each node n in a recursive manner as it is explained in definition 4.4.1.

Proposition 4.4.2: Let us assume the same conditions as in the previous definition, then for every node $n \in V(T)$, P_n is a measure on \mathcal{F}_X and therefore (X, \mathcal{F}_X, P_n) is a **measure space**.

Proof. We have:

- $0 \leq P_n(A) \leq \infty$ for any $A \in \mathcal{F}_X$ since $P_n(A) = \sum_{x \in A} \mu_i(x)$ and for all $i \in \{k \mid n_k \in V(T)\}$, $x \in X \subseteq \mathcal{X}$, it is fulfilled that $\mu_i(x) \in [0, 1]$ (see proposition 4.4.1).
- $P_n(\emptyset) = 0$ since $P_n(\emptyset) = \sum_{x \in \emptyset} \mu_n(x) = 0$

- If $A_j \in \mathcal{F}_X$, $j = 1, 2, \dots$ and the A_j 's are mutually disjoint, i.e., $A_j \cap A_k = \emptyset$ for any $j \neq k$, then $P_n\left(\bigcup_{j=1}^{\infty} A_j\right) = \sum_{j=1}^{\infty} P_n(A_j)$ since

$$\begin{aligned} P_n\left(\bigcup_{j=1}^{\infty} A_j\right) &= \sum_{x \in \bigcup_{j=1}^{\infty} A_j} \mu_n(x) \\ &\stackrel{A_j\text{'s mutually disjoint}}{=} \sum_{j=1}^{\infty} \left(\sum_{x \in A_j} \mu_n(x) \right) \\ &= \sum_{j=1}^{\infty} P_n(A_j) \end{aligned}$$

□

Definition 4.4.3: Let (X, \mathcal{F}_X, P_n) be the measure space defined above. Let $T = (V, E)$ be a SODT and let $A \in \mathcal{F}_X$ be a subset of the measurement space. We define the **fuzzy cardinality** of the set A at an arbitrary node $n_i \in V$ of the SODT as the P_{n_i} measure of A :

$$P_{n_i}(A) = \sum_{o \in A} \mu_i(o). \quad (4.3)$$

Thus, the fuzzy cardinality of A at n_i results from summing up the degrees of membership of each of its elements to this node n_i , which are described in definition 4.4.1.

Note: For simplification of notation we will call the P_{n_i} measure P_i measure.

P_n measure on \mathcal{B}_X , the Borel σ -algebra on X

In the case $X \subseteq \mathcal{X}$ is uncountable it is not possible to define a reasonable measure for every subset of X , i.e., it is not possible to consider the \mathcal{F}_X σ -field considered in the previous section for X finite (see [61]). This is why it is necessary to consider a σ -field which is smaller than the power set.

On the real line \mathbb{R} , there is a special σ -field which is often used. Let \mathcal{C} be the collection of all finite open intervals on \mathbb{R} . Then $\mathcal{B} = \sigma(\mathcal{C})$ (σ -field generated by \mathcal{C}) is the Borel σ -field. The elements of \mathcal{B} are called Borel sets. The Borel σ -field \mathcal{B}^h on the h -dimensional Euclidean space \mathbb{R}^h can be similarly defined. It can be shown that all intervals, open sets and closed sets are Borel sets (see [61]). Our measurement space \mathcal{X} , for example, is a Borel set (see section 4.2).

Let $X \subseteq \mathcal{X}$ be a Borel set, we consider the Borel σ -field on X , which is defined as $\mathcal{B}_X = \{X \cap B : B \in \mathcal{B}^h\}$. As means of proving the consistency of the measure defined in this section, we propose the following lemma.

Lemma 4.4.2: Let X, \mathcal{X} be two Borel sets on \mathbb{R}^h . If $X \subseteq \mathcal{X}$, then it is fulfilled that $\mathcal{B}_X \subseteq \mathcal{B}_{\mathcal{X}}$.

Proof. If $A \in \mathcal{B}_X$, then there exists a $B \in \mathcal{B}^h$ such that $A = X \cap B$, which can be written as $A = \mathcal{X} \cap X \cap B$.

By application of DeMorgan's law, $A = \mathcal{X} \cap (X^c \cup B^c)^c$ and, since \mathcal{B}^h is a σ -field and $X, B \in \mathcal{B}^h$, then $(X^c \cup B^c)^c \in \mathcal{B}^h$ which implies that $A = \mathcal{X} \cap (X^c \cup B^c)^c \in \mathcal{B}_X$. \square

Lemma 4.4.3: Let X be an uncountable Borel subset of our measurement space $\mathcal{X} \subseteq \mathbb{R}^h$ and let \mathcal{B}_X be the Borel σ -field on X , then (X, \mathcal{B}_X) is a **measurable space**.

Proof. It is trivial that \mathcal{B}_X fulfills the conditions of the definition of a σ -algebra (or σ -field).

1. $\emptyset \in \mathcal{B}_X$.
2. If $A \in \mathcal{B}_X$, then the complement $A^c \in \mathcal{B}_X$.
3. If $A_j \in \mathcal{B}_X$, $j = 1, 2, \dots$, then their union $\bigcup A_j \in \mathcal{B}_X$

\square

Let us define a measure on X .

Definition 4.4.4: Let T be a SODT and let (X, \mathcal{B}_X) be the measurable space defined above. For each node $n \in V(T)$ we define a set function on \mathcal{B}_X , $P_n : \mathcal{B}_X \longrightarrow \mathbb{R}$, as

$$P_n(A) = \int_A \mu_n(x) \lambda(dx)$$

for all $A \in \mathcal{B}_X$ (see definition 4.4.1) and λ the Lebesgue measure.

Notation: For simplicity of notation, we will write

$$P_n(A) = \int_A \mu_n(x) dx \quad \text{or} \quad P_n(A) = \int_A \mu_n d\lambda$$

Remark: For all $n \in V$, the function $\mu_n : \mathcal{X} \longrightarrow [0, 1]$ is a nonnegative (Borel) measurable function since it is continuous in the Borel set $\mathcal{X} \in \mathcal{B}_X$ (see proposition 1.4 in [61]). Thus, the integral $\int_A \mu_n(x) dx$ exists for all $A \in \mathcal{B}_X$ ([61], [58]). By application of lemma 4.4.2, the integral also exists for all $A \in \mathcal{B}_X$.

Moreover, P_n is absolutely continuous w.r.t. λ ($P_n \ll \lambda$) for λ the Lebesgue measure since $P_n(A) = 0$ implies that $\lambda(A) = 0$ ([61]).

In general, one can prove that for any nonnegative Borel function f defined in a measure space $(\Omega, \mathcal{F}, \lambda)$, the set function

$$\Lambda(A) = \int_A f d\lambda, \quad A \in \mathcal{F}$$

is a measure on (Ω, \mathcal{F}) . The function f is then called Radon-Nikodym *derivative* or *density* of Λ w.r.t. λ ([61]).

Proposition 4.4.3: Let us assume the same conditions as in the previous definition, then for every node $n \in V(T)$, P_n is a measure on \mathcal{B}_X and thus (X, \mathcal{B}_X, P_n) is a **measure space**.

Proof. We have:

- $0 \leq P_n(A) \leq \infty$ for any $A \in \mathcal{B}_X$ since $P_n(A) = \int_A \mu_i(x)dx$ and for all $i \in \{k \mid n_k \in V(T)\}$, $x \in \mathcal{X}$, it is fulfilled that $\mu_i(x) \in [0, 1]$ (see proposition 4.4.1).
- $P_n(\emptyset) = 0$ since $P_n(\emptyset) = \int_{\emptyset} \mu_n(x)dx = 0$
- If $A_j \in \mathcal{B}_X$, $j = 1, 2, \dots$ and A_j 's are mutually disjoint, i.e., $A_j \cap A_k = \emptyset$ for any $j \neq k$, then $P_n\left(\bigcup_{j=1}^{\infty} A_j\right) = \sum_{j=1}^{\infty} P_n(A_j)$ since

$$\begin{aligned}
 P_n\left(\bigcup_{j=1}^{\infty} A_j\right) & \stackrel{*}{=} \int_{\bigcup_{j=1}^{\infty} A_j} \mu_n(x)dx \\
 & \stackrel{A_j \text{'s mutually disjoint}}{=} \sum_{j=1}^{\infty} \left(\int_{A_j} \mu_n(x)dx \right) \\
 & = \sum_{j=1}^{\infty} P_n(A_j)
 \end{aligned}$$

* $\bigcup_{j=1}^{\infty} A_j \in \mathcal{B}_X$ and thus $\int_{\bigcup_{j=1}^{\infty} A_j} \mu_n(x)dx$ exists (see remark of definition 4.4.4). □

Definition 4.4.5: Let (X, \mathcal{B}_X, P_n) be the measure space defined above. Let $T = (V, E)$ be a SODT and let $A \in \mathcal{B}_X$ be a subset of the measurement space. We define the **fuzzy cardinality** of the set A at an arbitrary node $n_i \in V$ of the SODT as the P_{n_i} measure of A :

$$P_{n_i}(A) = \int_A \mu_i(x)dx. \quad (4.4)$$

Note: We have defined a measure of the set X of measurement vectors of any learning sample $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times \mathcal{C}$ at each node that is the analogue to the counting measure in the crisp case. Since we are using finite learning samples to learn our model, from now on, we will mostly consider the definitions of P_n measure and fuzzy cardinality given in the previous section for a finite subset of \mathcal{X} . Moreover, the P_n measure for the finite case is absolutely continuous w.r.t. the counting measure.

4.4.2 Properties of the degree of membership function μ

Election of a binary tree as topology for our model

The basic idea involved in SODT is to break up a complex decision into a union of several simpler decisions, being thus one of the well known *divide and conquer* methods. The

underlying topology of most of these methods, and of ours in concrete, is a *directed tree*. In order to see that the election of a *binary* tree, instead of a k -ary tree with $k > 2$, as graphical structure $T = (V, E)$ of our model (see chapter 2) is correct, we need the following section to see that the sum of the fuzzy cardinalities at both descendant nodes is equal to the fuzzy cardinality at the parent node.

Bipartition at each node

At each parent node of a crisp decision tree ⁵, the union of the disjoint sets of elements of the measurement space \mathcal{X} that go to the left and right child nodes respectively, is the set of elements belonging to the parent node, i.e., every splitting of a node n creates a partition of the original set of elements belonging to that node n (see section 3.4). In the SODT case, we do not have a crisp division of the elements of the measurement space going to the left child node and elements going to the right child node, but we can prove that the sum of the degrees of membership of an element $o \in \mathcal{X}$ to the left and to the right child nodes of any node $n \in Nodes(T)$ is equal to the degree of membership to the parent node n . And thus, the sum of the fuzzy cardinalities of the set \mathcal{X} at both successor nodes of any node $n \in V$ gives as result the fuzzy cardinality of the set \mathcal{X} at the original node n . The same can be proved for the input set X of a given learning sample \mathcal{L} . Let us see these assertions in detail in the following proposition and its corollaries:

Proposition 4.4.4: Let $T = (V, E)$ be a SODT labeled as we proposed in theorem 2.7.3, i.e. $E = \{(n_i, n_j) \in (V \setminus V_{leaves}) \times V \mid j = 2i \vee j = 2i + 1\}$. Let $n_i \in V \setminus V_{leaves}$ be a node of T with unpreciseness factor m_i associated and $\mu_i(o)$ the degree of membership of an element $o \in \mathcal{X}$ to this node n_i . Then the following property is fulfilled:

$$\mu_{2i}(o) + \mu_{2i+1}(o) = \mu_i(o) \quad (4.5)$$

i.e. the sum of the degrees of membership of an object $o \in \mathcal{X}$ to the right and left child nodes n_R and n_L of a node $n \in Nodes(T)$ gives the degree of membership to its parent node n , which means

$$\mu_{n_L}(o) + \mu_{n_R}(o) = \mu_n(o). \quad (4.6)$$

As an illustration see figures 4.6 and 4.7, where n_i or n are considered to be the root node and every $o \in \mathcal{X}$ belongs to the root node with degree of membership 1.

Proof. First of all let us clarify some values we have to know to calculate $\mu_{2i}(o)$ and $\mu_{2i+1}(o)$:

since $2i \equiv 0 \pmod{2}$, then, by equation (4.2) $P_{2i} := P(n_{2i}) = n_{\frac{2i}{2}} = n_i$

and since $2i + 1 \equiv 1 \pmod{2}$, then, by equation (4.2) $P_{2i+1} := P(n_{2i+1}) = n_{\frac{2i+1-1}{2}} = n_i$

and now, by applying lemma 4.3.1 we obtain

$$\begin{aligned} \mu_{2i}(o) + \mu_{2i+1}(o) &= FL(o_{q_i}, \sigma_i, m_i)\mu_i(o) + FG(o_{q_i}, \sigma_i, m_i)\mu_i(o) \\ &= \mu_i(o)(FL(o_{q_i}, \sigma_i, m_i) + FG(o_{q_i}, \sigma_i, m_i)) \\ &= \mu_i(o) \end{aligned}$$

⁵see chapter 3

□

Corollary 4.4.1: Let X be a finite subset of the measurement space $\mathcal{X} \subseteq \mathbb{R}^h$. Let $T = (V, E)$ be a SODT and let $P_i(A)$ be the fuzzy cardinality of a set $A \in \mathcal{F}_X$ at node $n_i \in V$. Then the following property is fulfilled:

$$P_{2i}(A) + P_{2i+1}(A) = P_i(A) \quad (4.7)$$

i.e., the sum of the fuzzy cardinalities of a set $A \subseteq X$ at right and left child nodes of a node $n \in Nodes(T)$ gives the fuzzy cardinality at their parent node n . Obviously, it is then fulfilled that for our given finite learning sample $\mathcal{L} = (X, Y) \in \mathcal{X} \times \mathcal{C}$,

$$P_{2i}(X) + P_{2i+1}(X) = P_i(X).$$

Proof. It is trivial by applying proposition 4.4.4:

$$\begin{aligned} P_i(A) &= \sum_{o \in A} \mu_i(o) \\ &\stackrel{Prop. 4.4.4}{=} \sum_{o \in A} (\mu_{2i}(o) + \mu_{2i+1}(o)) \\ &= \sum_{o \in A} \mu_{2i}(o) + \sum_{o \in A} \mu_{2i+1}(o) \\ &\stackrel{Def. 4.4.3}{=} P_{2i}(A) + P_{2i+1}(A) \end{aligned}$$

□

Let us generalize this corollary for the complete measurement space \mathcal{X} :

Corollary 4.4.2: Let X be a Borel subset of \mathcal{X} . Let $T = (V, E)$ be a SODT and let $P_i(B)$ be the fuzzy cardinality of a set $B \in \mathcal{B}_X$ at node $n_i \in V$. Then, the following property is fulfilled:

$$P_{2i}(B) + P_{2i+1}(B) = P_i(B) \quad (4.8)$$

i.e., the sum of the fuzzy cardinalities of a set $B \in \mathcal{B}_X$ at right and left successors of a node $n \in Nodes(T)$ gives the fuzzy cardinality at their parent node n . Since \mathcal{X} itself is a Borel set and obviously, $\mathcal{X} \in \mathcal{B}_X$, it is then fulfilled that

$$P_{2i}(\mathcal{X}) + P_{2i+1}(\mathcal{X}) = P_i(\mathcal{X}).$$

Proof. The proof is similar to the proof of the previous corollary for the finite case.

$$\begin{aligned} P_i(A) &= \int_A \mu_i(x) dx \\ &\stackrel{Prop. 4.4.4}{=} \int_A (\mu_{2i}(x) + \mu_{2i+1}(x)) \\ &= \int_A \mu_{2i}(x) + \int_A \mu_{2i+1}(x) \\ &\stackrel{Def. 4.4.5}{=} P_{2i}(A) + P_{2i+1}(A) \end{aligned}$$

□

Fuzzy partition of the measurement space \mathcal{X}

Recall that in every crisp decision tree, the sets of elements belonging to the terminal nodes form a partition of the measurement space \mathcal{X} . In the SODT case this partition in the usual sense is not possible since the fact whether the elements belong to or do not belong to a certain set can not be determined. But they do it with certain degrees of membership.

Ruspini ([59]) extended the notion of partition of a measurement space to fuzzy sets (see appendix A) as follows:

Definition 4.4.6: The set of fuzzy sets $\{A_1, \dots, A_n\}$ form a partition of the measurement space \mathcal{X} iff

$$\forall x \in \mathcal{X} \quad \sum_{i=1}^n \mu_{A_i}(x) = 1 \quad (4.9)$$

The essential requirement, then is, that the sum of the degrees of membership of an element of the measurement space over the partition is one. Furthermore, notice that if A_1, \dots, A_n are restricted to crisp sets then this corresponds to the standard definition of a partition of \mathcal{X} .

Let us see in the following lemma that the leaf nodes of a SODT, considered as fuzzy sets, form a fuzzy partition of the measurement space:

Lemma 4.4.4: Let $T = (V, E)$ be a SODT and let $V_{leaves} \subseteq V$ be the subset of the set of nodes of T with degree 1 (see definition 2.7.2). Then, for all measurement vectors $o \in \mathcal{X}$

$$\sum_{i \in V_{leaves}} \mu_i(o) = \mu_1(o) = 1 \quad (4.10)$$

i.e., the sum of the degrees of membership of an element $o \in \mathcal{X}$ to all the leaf nodes is equal to the degree of membership to the root node, which is defined by assumption as 1, as it is explained in remark of definition 4.4.1.

Proof. By induction on the number $t = \#V_{leaves} \in \mathbb{N}$ of terminal nodes.

If $\#V_{leaves} = 1$, then the unique terminal node n and root node coincide ⁶ and the lemma is obviously fulfilled.

We suppose the lemma to be valid for a SODT $T = (V, E)$ of order $|T| = t$ ⁷ and terminal nodes set called V_{leaves} , then

$$\sum_{i \in V_{leaves}} \mu_i(o) = \mu_1(o).$$

⁶By application of "every tree has at least one root node and one leaf node"

⁷By definition 2.2.1, $|T|$ is equal to the number of vertices $\#V$ of the tree

Let us increase the order of T , $|T|$, by splitting one $n_j \in V_{leaves}$ into the corresponding n_{2j} and n_{2j+1} child nodes. The new SODT $T' = (V', E')$ nodes and leaves sets are:

$$Nodes(T') = Nodes(T) \cup \{n_{2j}, n_{2j+1}\}$$

$$V'_{leaves} = (V_{leaves} \setminus \{n_j\}) \cup \{n_{2j}, n_{2j+1}\}$$

and thus $\#V'_{leaves} = \#V_{leaves} + 1$. Let us see if the claim is also valid for T' .

By application of proposition 4.4.4 ($\mu_{2i}(o) + \mu_{2i+1}(o) = \mu_i(o)$) it follows:

$$\begin{aligned} \sum_{i \in V'_{leaves}} \mu_i(o) &= \sum_{i \in (V_{leaves} \setminus \{n_j\}) \cup \{n_{2j}, n_{2j+1}\}} \mu_i(o) \\ &= \sum_{i \in (V_{leaves} \setminus \{n_j\})} \mu_i(o) + \sum_{i \in \{n_{2j}, n_{2j+1}\}} \mu_i(o) \\ &= \sum_{i \in (V_{leaves} \setminus \{n_j\})} \mu_i(o) + \mu_{2j}(o) + \mu_{2j+1}(o) \\ &= \sum_{i \in (V_{leaves} \setminus \{n_j\})} \mu_i(o) + \mu_j(o) \\ &= \sum_{i \in V_{leaves}} \mu_i(o) \\ &= \mu_1(o) \quad \text{by hypothesis of induction.} \end{aligned}$$

□

Proposition 4.4.5: Let X be a finite subset of our measurement space \mathcal{X} . Let us assume the conditions of the previous lemma to be fulfilled and let $A \in \mathcal{F}_X$ be an element of the σ -algebra $\mathcal{F}_X \equiv$ a measurable set. Then, the sum of the fuzzy cardinalities of the set $A \subseteq X$ at all leaves nodes is equal to the number of elements of A .

$$\sum_{i \in V_{leaves}} P_i(A) = \#A \quad (4.11)$$

Proof. Let us start by applying definition 4.4.3 to finally apply lemma 4.4.4

$$\begin{aligned} \sum_{i \in V_{leaves}} P_i(A) &= \sum_{i \in V_{leaves}} \left(\sum_{o \in A} \mu_i(o) \right) \\ &= \sum_{o \in A} \sum_{i \in V_{leaves}} \mu_i(o) \\ &\stackrel{\text{Lemma 4.4.4}}{=} \sum_{o \in A} \mu_1(o) \\ &= \sum_{o \in A} 1 = \#A \end{aligned}$$

The assumption that $\mu_1(o) = 1$ is explained in the remark of definition 4.4.1.

□

Remark: Obviously, the previous proposition is also valid for the measurements set of our given finite learning sample $\mathcal{L} = (X, Y) \in \mathcal{X} \times \mathcal{C}$

$$\sum_{i \in V_{leaves}} P_i(X) = \#X$$

which suggests that if we would consider the nodes of the SODT as fuzzy sets, then the terminal subsets would form a partition of X .

Proposition 4.4.6: Let X be a Borel subset of our measurement space \mathcal{X} . Let us assume the conditions of the previous lemma to be fulfilled and let $B \in \mathcal{B}_{\mathcal{X}}$ be an element of the σ -algebra $\mathcal{B}_{\mathcal{X}} \equiv$ a measurable set. Then, the sum of the fuzzy cardinalities of the set $B \in \mathcal{B}_{\mathcal{X}}$ at all leaves nodes is equal to the Lebesgue measure λ of B .

$$\sum_{i \in V_{leaves}} P_i(B) = \lambda(B) \quad (4.12)$$

Proof. In a similar way we preceeded in the previous proposition, here we start by applying definition 4.4.5 to finally apply lemma 4.4.4

$$\begin{aligned} \sum_{i \in V_{leaves}} P_i(B) &= \sum_{i \in V_{leaves}} \left(\int_A \mu_i(x) dx \right) \\ &= \int_A \sum_{i \in V_{leaves}} \mu_i(x) dx \\ &\stackrel{\text{Lemma 4.4.4}}{=} \int_A \mu_1(x) dx \\ &= \int_A 1 dx = \lambda(A) \end{aligned}$$

□

The previous result is also valid for the measurement space \mathcal{X} and thus

$$\sum_{i \in V_{leaves}} P_i(\mathcal{X}) = \lambda(\mathcal{X})$$

which suggests that if we would consider the nodes of the SODT as fuzzy sets, then the terminal subsets would form a partition of \mathcal{X} .

Claim: Let $T = (V, E)$ be a SODT. Considering the definition of union of SODT nodes analogously to the union of fuzzy sets proposed by Ernest Czogała and Jacek Łęski also in [13] (see appendix A), we could say that the union of the leaves of SODT is equal to the measure of our \mathcal{X} .

In order to show this claim, let $n_i, n_j \in V$ and let $(n_i \cup_S n_j)$ be defined as in appendix A.1:

$$(n_i \cup_S n_j)(x) = S(n_i(x), n_j(x)) = S(\mu_i(x), \mu_j(x)).$$

If we choose the t-conorm S to be the bounded sum (see figure 4.8)

$$S(x, y) = \min(x + y, 1),$$

We define the union of two nodes $n_i, n_j \in V$ at a finite subset X of \mathcal{X} as

$$(n_i \tilde{\cup} n_j)(X) := \sum_{x \in X} (n_i \cup_S n_j)(x)$$

. Let us see then that

$$\bigcup_{i \in V_{leaves}} n_i(X) = \nu(X),$$

where ν is any measure defined on the σ -algebra \mathcal{F}_X (the power set of X), as for example the cardinality of a set.

For all $i, j \in V_{leaves}$

$$(n_i \cup_S n_j)(x) = S(n_i(x), n_j(x)) = S(\mu_i(x), \mu_j(x)).$$

where, in this case, S could be substituted by

$$S'(x, y) = x + y$$

since $\mu_i(x) + \mu_j(x) \leq 1$ for all $x \in X$ and $i, j \in V_{leaves}$ as proved in lemma 4.4.4 $\left(\sum_{i \in V_{leaves}} \mu_i(o) = 1 \text{ and } \mu_i \geq 0 \forall i \in V \right)$ and thus

$$(n_i \cup_S n_j)(x) = S'(\mu_i(x), \mu_j(x)) = \mu_i(x) + \mu_j(x)$$

$$\begin{aligned} \bigcup_{i \in V_{leaves}} n_i &= \sum_{x \in X} \sum_{i \in V_{leaves}} \mu_i(x) \\ &\stackrel{\text{lemma 4.4.4}}{=} \sum_{x \in X} \mu_1(x) = \sum_{x \in X} 1 = \#X. \end{aligned}$$

For X a Borel subset of \mathcal{X} , it can be analogously seen that

$$\bigcup_{i \in V_{leaves}} n_i(X) = \lambda(X),$$

for λ the Lebesgue measure on \mathcal{B}_X and, since \mathcal{X} is a Borel set, then the union of the leaves of a SODT at \mathcal{X} , considered as fuzzy sets, is equal to the measure of \mathcal{X} .

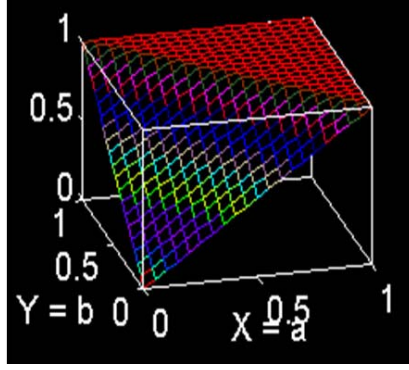


Figure 4.8: Bounded sum fuzzy operator

Asymptotical behaviour of degree of membership function μ

Proposition 4.4.7: Let \mathcal{X} be a measurement space (see definition 3.3.1), T a SODT and (q, σ, m) the 3-tuple associated to an arbitrary node $n \in Nodes(T)$. If we denote the left and right child nodes of n by n_L and n_R respectively, then for all $o \in \mathcal{X}$

$$\lim_{m \rightarrow 0} \mu_{n_L}(o) = \lim_{m \rightarrow 0} \mu_{n_R}(o) = \frac{\mu_n(o)}{2}$$

which means that in this case, the splitting of the space is completely diffuse, there is no real separation because all the elements will belong to the left and right child nodes with the same degree of membership.

In the particular case of n being the root node n_1 , then if m_1 tends to 0, it follows

$$\mu_{n_2}(o) = \mu_{n_3}(o) = 0.5$$

Proof.

$$\begin{aligned} \lim_{m \rightarrow 0} \mu_{n_L}(o) &= \lim_{m \rightarrow 0} FL(o_q, \sigma, m) \cdot \mu_n(o) \\ &= \lim_{m \rightarrow 0} \frac{\tanh((\sigma - o_q) \cdot m) + 1}{2} \cdot \mu_n(o) \\ &= \lim_{m \rightarrow 0} \frac{\frac{e^{(\sigma - o_q) \cdot m} - e^{-(\sigma - o_q) \cdot m}}{e^{(\sigma - o_q) \cdot m} + e^{-(\sigma - o_q) \cdot m}} + 1}{2} \cdot \mu_n(o) \\ &= \frac{1}{2} \cdot \mu_n(o) \end{aligned}$$

and analogously

$$\begin{aligned}
 \lim_{m \rightarrow 0} \mu_{n_R}(o) &= \lim_{m \rightarrow 0} FG(o_q, \sigma, m) \cdot \mu_n(o) \\
 &= \lim_{m \rightarrow 0} \frac{\tanh((o_q - \sigma) \cdot m) + 1}{2} \cdot \mu_n(o) \\
 &= \lim_{m \rightarrow 0} \frac{\frac{e^{-(\sigma - o_q) \cdot m} - e^{(\sigma - o_q) \cdot m}}{e^{-(\sigma - o_q) \cdot m} + e^{(\sigma - o_q) \cdot m}} + 1}{2} \cdot \mu_n(o) \\
 &= \frac{1}{2} \cdot \mu_n(o)
 \end{aligned}$$

□

Proposition 4.4.8: Let us assume the same conditions as in proposition 4.4.7 and let us call n_P the parent node of $n \in Nodes(T)$. Then

$$\mu_n(\sigma_P) = 0.5 \mu_{n_P}(\sigma_P)$$

being σ_P the second component of the 3-tuple associated to node n_P . In other words, the degree of membership of the threshold value σ to the right and left node is the same and in the particular case that n_P is the root node, then this degree of membership is equal to 0.5.

Proof. Let us suppose a labelled SODT, then there exists a $k \in 1, \dots, 2^L - 1$, being L the level of the SODT, such that $n \in Nodes(T)$ can be labelled as $n_k \in Nodes(T)$. Then, we can distinguish between two cases:

- case 1: $k \equiv 0 \pmod{2}$, then

$$\begin{aligned}
 \mu_{n_k}(\sigma_{P_k}) &:= \mu_k(\sigma) = FL(\sigma_{P_k}, \sigma_{P_k}, m_{P_k}) \cdot \mu_{P_k}(\sigma) \\
 &= \frac{\tanh((\sigma_{P_k} - \sigma_{P_k}) \cdot m) + 1}{2} \cdot \mu_{P_k}(\sigma) \\
 &= \frac{1}{2} \mu_{P_k}(\sigma_{P_k})
 \end{aligned}$$

- case 2: $k \equiv 1 \pmod{2}$, then

$$\begin{aligned}
 \mu_{n_k}(\sigma_{P_k}) &:= \mu_k(\sigma) = FG(\sigma_{P_k}, \sigma_{P_k}, m_{P_k}) \cdot \mu_{P_k}(\sigma) \\
 &= \frac{\tanh((\sigma_{P_k} - \sigma_{P_k}) \cdot m) + 1}{2} \cdot \mu_{P_k}(\sigma) \\
 &= \frac{1}{2} \mu_{P_k}(\sigma_{P_k})
 \end{aligned}$$

□

4.5 Guidelines for the learning and inference process of a SODT

The application of SODT to classification can be separated into two subtasks:

- SODT learning (training) (see sections 4.6 - 4.10)
- SODT inference to classify instances (see section 4.11).

The general guidelines for these two procedures can be seen in figure 4.9. In this figure we can see that a learning sample \mathcal{L} is required as input for the SODT learning process. Indeed, this process is divided into three main procedures, SODT induction, SODT stopping rule and SODT terminal node class assignment, which are briefly explained in the next section and individually detailed in sections 4.7, 4.8 and 4.9, respectively. As result of this learning process, a SODT is obtained, which can be used as usual classifier or as fuzzy classifier, as it is explained in section 4.11. According to this election, a class or a vector with degrees of membership to classes will be assigned to any new instance of the measurement space \mathcal{X} .

4.6 Construction of the SODT

Analogously as in the crisp DT case, the entire construction of the SODT is also divided into three tasks:

1. The selection of the variable or feature axis q and threshold point σ for this variable q to split in every internal node, as well as the selection of the unpreciseness factor m , which can be chosen individually for each node or as a global value for all nodes of the SODT. However, it makes more sense to study the case it is chosen node-specific, because the unpreciseness or slope of the operators should be chosen according to the domain of the variable $x_q \in \mathcal{X}_q$ selected for splitting the measurement space \mathcal{X} . Due to the similarity of this process with decision tree induction, we call it **SODT induction**.
2. The decisions when to declare a node terminal or to continue splitting it, which can be summarized in finding a stopping rule.
3. The assignment of a class to each of the terminal nodes.

Let us see them in details in next sections 4.7, 4.8 and 4.9.

4.7 SODT induction

SODT, as all tree structured classifiers, is constructed by repeated splits of subsets of \mathcal{X} into two descendant subsets, beginning with \mathcal{X} itself. The fundamental idea is also here

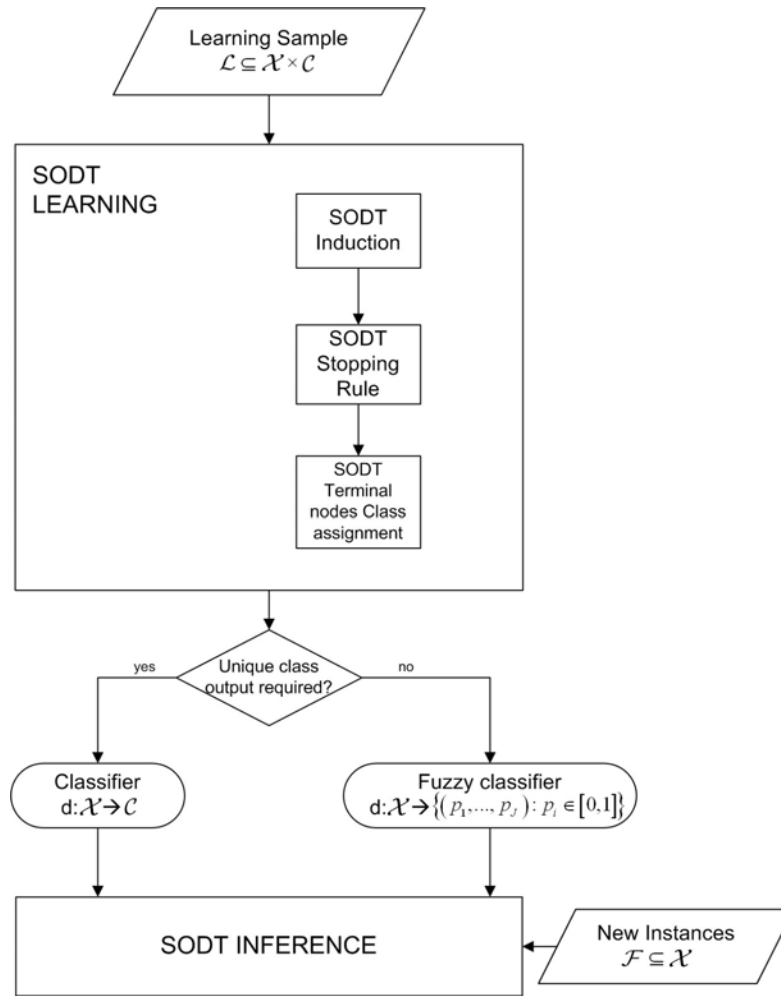


Figure 4.9: Outline of the learning and inference processes of a SODT

the selection of each node variable, threshold and unpreciseness factor used to split, so that the data in each of the descendant subsets are “purer” than the data in the parent set.

As we already commented in 3.6, for ordered or numerical variables the natural splits are binary of the form $x \leq \sigma$, and this applies to both continuous measurements and to ordered categories. In our method, we consider the binary split of the form $x <^{FL} \sigma$. It will also result in binary trees because $FL(x, y, z) + FG(x, y, z) = 1$ (see lemma 4.3.1).

4.7.1 Function $\rho_m(j|n)$ of probability

In SODT case, we do not interpret probabilities in the frequentist way, but in terms of degree of membership.

Claim: Let $\mathcal{C} = \{1, \dots, J\}$ be a set of classes and let us call $\mathcal{F}_{\mathcal{C}}$ the power set of \mathcal{C} , which is the largest σ -algebra on \mathcal{C} (trivial, analogously to lemma 4.4.1). Thus, $(\mathcal{C}, \mathcal{F}_{\mathcal{C}})$ is a **measurable space** and we can define a probability measure on \mathcal{C} .

Definition 4.7.1: Let $(\mathcal{C}, \mathcal{F}_{\mathcal{C}})$ be the measurable space defined above. Let $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times \mathcal{C}$ be a learning data set and T a SODT. For all $n \in V(T)$, we define a set function $\Psi : \mathcal{F}_{\mathcal{C}} \rightarrow [0, 1]$ such that for all $C \in \mathcal{F}_{\mathcal{C}}$

$$\Psi(C) = \frac{P_n(X^C)}{P_n(X)}$$

where $P_n(A)$ is the fuzzy cardinality of a set $A \in \mathcal{F}_X$ at node n (see definition 4.4.3) and $X^C := \{o \in X \mid (o, y) \in \mathcal{L} \text{ for a } y \in C\}$. Thus, since \mathcal{L} is finite,

$$\Psi(C) = \frac{\sum_{\substack{(o,j) \in \mathcal{L} \\ j \in C}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)}.$$

Lemma 4.7.1: Let us suppose that the same conditions as in definition 4.7.1 are fulfilled. Then, the set function $\Psi : \mathcal{F}_{\mathcal{C}} \rightarrow [0, 1]$ is a measure and thus $(\mathcal{C}, \mathcal{F}_{\mathcal{C}}, \Psi)$ is a **measure space**.

Proof. • $0 \leq \Psi(C)$ for any $C \in \mathcal{F}_{\mathcal{C}}$ since

$$\Psi(C) = \frac{\sum_{\substack{(o,j) \in \mathcal{L} \\ j \in C}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)}$$

and for all $n \in V(T)$, $o \in \mathcal{X}$, it is fulfilled that $\mu_n(o) \in [0, 1]$ (see proposition 4.4.1).

- $\Psi(\emptyset) = 0$. Trivial.

- If $C_i \in \mathcal{F}_C$, $i = 1, 2, \dots$ and C_i 's are mutually disjoint, i.e., $C_i \cap C_k = \emptyset$ for any $i \neq k$, then $\Psi\left(\bigcup_{i=1}^{\infty} C_i\right) = \sum_{i=1}^{\infty} \Psi(C_i)$ since

$$\begin{aligned}
\Psi\left(\bigcup_{i=1}^{\infty} C_i\right) &= \frac{\sum_{\substack{(o,j) \in \mathcal{L} \\ j \in \bigcup_{i=1}^{\infty} C_i}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} \\
&\stackrel{C_i \text{ disjoint}}{=} \frac{\sum_{i=1}^{\infty} \sum_{\substack{(o,j) \in \mathcal{L} \\ j \in C_i}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} \\
&= \sum_{i=1}^{\infty} \frac{\sum_{\substack{(o,j) \in \mathcal{L} \\ j \in C_i}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} \\
&= \sum_{i=1}^{\infty} \Psi(C_i)
\end{aligned}$$

□

Proposition 4.7.1: Let us suppose that the same conditions as in definition 4.7.1 are fulfilled. Then, the set function $\Psi : \mathcal{F}_C \longrightarrow [0, 1]$ is a **probability measure** and thus $(\mathcal{C}, \mathcal{F}_C, \Psi)$ is a **probability space**.

Proof. By lemma 4.7.1, $(\mathcal{C}, \mathcal{F}_C, \Psi)$ is a measure space, then to prove that it is also a probability space we just need to see the additional condition that $\Psi(\mathcal{C}) = 1$:

$$\begin{aligned}
\Psi(\mathcal{C}) &= \frac{\sum_{\substack{(o,j) \in \mathcal{L} \\ j \in \mathcal{C}}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} \\
&\stackrel{\mathcal{C}=\{1,\dots,J\}}{=} \frac{\sum_{j=1}^J \sum_{(o,j) \in \mathcal{L}} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} \\
&\stackrel{\mathcal{L}=(X,Y) \subseteq \mathcal{X} \times \mathcal{C}}{=} \frac{\sum_{o \in X} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} = 1
\end{aligned}$$

□

Definition 4.7.2: Let $(\mathcal{C}, \mathcal{F}_C, \Psi)$ be the probability space defined in proposition 4.7.1 for a node $n \in V(T)$. Let $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times \mathcal{C}$ be a learning data set and T a SODT. For all $n \in V(T)$, we define the probability of a class $j \in \mathcal{C} = \{1, \dots, J\}$ in a SODT node n as the function $\rho_m : \mathcal{C} \longrightarrow [0, 1]$, being m the unpreciseness factor associated to the predecessor of n , $P(n)$, defined by

$$\rho_m(j|n) = \Psi(\{j\}) \quad \forall j \in \mathcal{C}$$

where $\Psi : \mathcal{F}_C \longrightarrow [0, 1]$ is the probability function defined above for each fixed node $n \in V(T)$. Thus, $\rho_m(j|n)$ can also be written as

$$\rho_m(j|n) = \frac{P_n(X^j)}{P_n(X)} \quad \forall j \in \mathcal{C}$$

where

$$X^j := X^{\{j\}} = \{o \in X \mid (o, j) \in \mathcal{L}\}.$$

Let us see that the definition of $\rho_m(j|n)$ fulfills the requirements of the definition of impurity function $((p_1, \dots, p_J)$ satisfying $p_j \geq 0$, $j = 1, \dots, J$, $\sum_j p_j = 1$) with the following proposition.

Proposition 4.7.2: Given $\mathcal{C} = \{1, \dots, J\}$, a SODT T , an arbitrary $n \in V(T)$ and ρ_m defined as above, then $(\rho_m(1|n), \dots, \rho_m(J|n)) \in \mathbb{R}^J$ is an element of the standard (topological) $(J - 1)$ -simplex Δ^{J-1} .

Proof. Let us see that $(\rho_m(1|n), \dots, \rho_m(J|n))$ fulfills the requirements exposed in section 3.6.1 for an element of the standard (topological) $(J - 1)$ -simplex.

- $\rho_m(j|n) \geq 0 \forall j = 1, \dots, J$:

Trivial since $\rho_m(j|n) = \frac{\sum_{(o,j) \in \mathcal{L}} \mu_n(o, m)}{\sum_{o \in X} \mu_n(o, m)}$ and the range of μ_n for all $n \in T$ is the interval $[0, 1]$ (see definition 4.4.1).

- $\sum_{j=1}^J \rho_m(j|n) = 1$:

$$\begin{aligned} \sum_{j=1}^J \rho_m(j|n) &= \sum_{j=1}^J \frac{\sum_{(o,j) \in \mathcal{L}} \mu_n(o, m)}{\sum_{o \in X} \mu_n(o, m)} \\ &= \frac{1}{\sum_{o \in X} \mu_n(o, m)} \sum_{j=1}^J \sum_{(o,j) \in \mathcal{L}} \mu_n(o, m) \\ &\stackrel{*}{=} \frac{1}{\sum_{o \in X} \mu_n(o, m)} \sum_{o \in X} \mu_n(o, m) = 1. \end{aligned}$$

* Let us see the development of the last step:

$$\begin{aligned} \sum_{j=1}^J \sum_{(o,j) \in \mathcal{L}} \mu_n(o, m) &= \sum_{(o,1) \in \mathcal{L}} \mu_n(o, m) + \dots + \sum_{(o,J) \in \mathcal{L}} \mu_n(o, m) \\ &= \sum_{o \in X} \mu_n(o, m) \end{aligned}$$

since for all $o \in \mathcal{X}$, there exists exactly one $j \in \mathcal{C} = \{1, \dots, J\}$ such that $(o, j) \in \mathcal{L}$.

□

4.7.2 Proportion $\rho_{m,n}$ of objects going to node n

Definition 4.7.3: Let T be a SODT and $n_i \in \text{Nodes}(T)$ with 3-tuple (q_i, σ_i, m_i) associated and parent node n with associated 3-tuple (q, σ, m) . Then, we define the proportion of objects ρ_{m,n_i} of the learning set $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times \mathcal{C}$ going to node n_i in the following way:

$$\begin{aligned} \rho_{m,n_i} &:= \frac{P_{n_i}(X)}{P_n(X)} \\ &= \frac{\sum_{o \in X} \mu_{n_i}(o, m)}{\sum_{o \in X} \mu_n(o, m_P)} \end{aligned}$$

where m_P is the third coordinate of the 3-tuple associated to the direct predecessor of n , $P(n)$ (see definition 2.6.2). In other words, m_P is the unpreciseness factor associated to the parent node of n .

Remark: This definition transformed in terms of left and right child nodes yields the following:

Definition 4.7.4: Let T be a SODT and $t \in \text{Nodes}(T)$ with associated 3-tuple (q, σ, m) . Let us consider the parent node of t , t_P , with associated 3-tuple (q_P, σ_P, m_P) , and let the left and right child nodes of t be t_L and t_R respectively. The proportion of objects $\rho_{m,L}$ of $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times \mathcal{C}$ going to the left child node of t is defined as:

$$\rho_{m,L} := \frac{P_{t_L}(X)}{P_t(X)},$$

which means

$$\rho_{m,L} := \frac{\sum_{o \in X} \mu_{t_L}(o, m)}{\sum_{o \in X} \mu_t(o, m_P)} = \frac{\sum_{o \in X} \frac{\tanh\left(\frac{(\sigma - o) \cdot m}{2}\right) + 1}{2} \cdot \mu_t(o, m)}{\sum_{o \in X} \mu_t(o, m_P)}$$

and analogously, the proportion of objects $\rho_{m,R}$ going to the right child node of t is:

$$\rho_{m,R} := \frac{P_{t_R}(X)}{P_t(X)},$$

which means

$$\rho_{m,R} := \frac{\sum_{o \in X} \mu_{t_R}(o, m)}{\sum_{o \in X} \mu_t(o, m_P)} = \frac{\sum_{o \in X} \frac{\tanh\left(\frac{(o - \sigma) \cdot m}{2}\right) + 1}{2} \cdot \mu_t(o, m_P)}{\sum_{o \in X} \mu_t(o, m_P)}.$$

Proposition 4.7.3: Let $T = (V, E)$ be a SODT labeled as we proposed in theorem 2.7.3, i.e. $E = \{(n_i, n_j) \in (V \setminus V_{leaves}) \times V \mid j = 2i \vee j = 2i + 1\}$ and let $n_i \in V \setminus V_{leaves}$ be a node of T with unpreciseness factor m associated. Then,

$$\rho_{m,n_{2i}} + \rho_{m,n_{2i+1}} = 1$$

Proof.

$$\begin{aligned}
\rho_{m,n_{2i}} + \rho_{m,n_{2i+1}} &= \frac{\sum_{o \in X} \mu_{n_{2i}}(o, m) + \sum_{o \in X} \mu_{n_{2i+1}}(o, m)}{\sum_{o \in X} \mu_{n_i}(o, m)} \\
&= \frac{\sum_{o \in X} (\mu_{n_{2i}}(o, m) + \mu_{n_{2i+1}}(o, m))}{\sum_{o \in X} \mu_{n_i}(o, m)} \\
&\stackrel{\text{Prop. 4.4.4}}{=} \frac{\sum_{o \in X} \mu_{n_i}(o, m)}{\sum_{o \in X} \mu_{n_i}(o, m)} = 1
\end{aligned}$$

□

4.7.3 SODT node impurity function

Definition 4.7.5: Given an impurity function $\Phi : \Delta^{(J-1)} \longrightarrow \mathbb{R}$ (see definition 3.6.2), and the probability measure $\rho_m : \mathcal{C} \longrightarrow [0, 1]$ for all $n \in \text{Nodes}(T)$, being $T = (V, E)$ a SODT, we define the **SODT node impurity function**

$$\tau : \text{Nodes}(T) \times (0, +\infty) \longrightarrow \mathbb{R}$$

of any node n as

$$\tau(n, m) = \Phi(\rho_m(1|n), \rho_m(2|n), \dots, \rho_m(J|n)) \quad (4.13)$$

where Φ is a function that fulfills the conditions required for impurity functions in definition 3.6.2.

Remark: Since $(\rho_m(1|n), \rho_m(2|n), \dots, \rho_m(J|n))$ is an element of the standard (topological) $(J-1)$ -simplex Δ^{J-1} (see proposition 4.7.2), we can express the SODT node impurity function as

$$\tau(n, m) = \Phi(\rho_m(1|n), \rho_m(2|n), \dots, \rho_m(J-1|n))$$

due to the fact that $\rho_m(J|n) = 1 - \sum_{j=1}^{J-1} \rho_m(j|n)$.

Often, the case $J = 2$ is studied. In this case, the SODT node impurity function can be expressed as

$$\tau(n, m) = \Phi(\rho_m(c|n)), \quad (4.14)$$

being c one of the two possible outcomes.

SODT Gini Index

In our case, the traditional Gini Index definition exposed in section 3.7

$$i : \text{Nodes}(T) \longrightarrow \mathbb{R}$$

$$i_G(n) = \sum_{i \neq j} p(i|n)p(j|n)$$

is modified:

$$\tau_{Gini} : Nodes(T) \times (0, +\infty) \longrightarrow \mathbb{R}$$

$$\tau_{Gini}(n, m) = \sum_{i \neq j} \rho_m(i|n)\rho_m(j|n) \quad (4.15)$$

where m is the unpreciseness factor associated to $P(n)$. This τ_{Gini} can also be written as

$$\tau_{Gini}(n, m) = \sum_j \rho_m(j|n) - \sum_j \rho_m^2(j|n) = 1 - \sum_j \rho_m^2(j|n).$$

In the two-class problem, the SODT Gini index reduces to

$$\tau(n, m) = 2\rho_m(1|n)\rho_m(2|n)$$

Remark: SODT Information Gain and SODT MR index are analogously computed based on the Information Gain and MR index definitions of sections 3.8 and 3.9.

4.7.4 Function ν of goodness

Definition 4.7.6: Let $o \in \mathcal{X}$ be the input of an element (o, c) of the learning data set \mathcal{L} and let n_i be a node of the SODT T , for $i = 1, \dots, 2^L - 1$, where L denotes the level of T . At each node n_k , we define the SODT *goodness* ν of the possible 3-tupla (q_k, σ_k, m_k) that can be associated to n_k , as the decrease in τ -impurity

$$\nu : (\{1, \dots, h\} \times \mathcal{X} \times (0, +\infty)) \times V(T) \longrightarrow \mathbb{R}$$

$\nu((q_k, \sigma_k, m_k), n_k) = \Delta\tau(n_k, m_{P_k}) = \tau(n_k, m_{P_k}) - \rho_{m_k, n_{2k}}\tau(n_{2k}, m_k) - \rho_{m_k, n_{2k+1}}\tau(n_{2k+1}, m_k)$ where $P_k := \{k \in \{1, \dots, 2^L - 1\} \mid n_{P_k} \in P(n_k)\}$, i.e. P_k is the label of the predecessor or parent of n_k . If we concentrate in one node n and want to see this definition in terms of right and left childs, it would be:

$$\nu : (\{1, \dots, h\} \times \mathcal{X} \times (0, +\infty)) \times V(T) \longrightarrow \mathbb{R}$$

$$\nu((q, \sigma, m), n) = \Delta\tau(n, m_P) = \tau(n, m_P) - \rho_{m, n_L}\tau(n_L, m) - \rho_{m, n_R}\tau(n_R, m)$$

where n_L and n_R represent the left and right child nodes of n respectively.

Theorem 4.7.1: ⁸ Let $T = (V, E)$ be a SODT and let $n_k, n_{2k}, n_{2k+1} \in Nodes(T)$ where m_k is the unpreciseness factor associated with node n_k , and m_{P_k} the one associated with the predecessor of n_k . Then,

$$\rho_{m_k, n_{2k}}\rho_{m_k}(c|n_{2k}) + \rho_{m_k, n_{2k+1}}\rho_{m_k}(c|n_{2k+1}) = \rho_{m_{P_k}}(c|n_k),$$

being $\rho_m(j|n)$ and $\rho_{m,n}$ the functions described in definitions 4.7.2 and 4.7.3, respectively.

⁸This theorem is the analogue to the *total probability theorem*, but in the SODT case

Proof. By definitions 4.7.2 and 4.7.3

$$\begin{aligned}
& \rho_{m_k, n_{2k}} \rho_{m_k}(c|n_{2k}) + \rho_{m_k, n_{2k+1}} \rho_{m_k}(c|n_{2k+1}) = \\
& \frac{P_{n_{2k}}(X)}{P_{n_k}(X)} \cdot \frac{P_{n_{2k}}(X^c)}{P_{n_{2k}}(X)} + \frac{P_{n_{2k+1}}(X)}{P_{n_k}(X)} \cdot \frac{P_{n_{2k+1}}(X^c)}{P_{n_{2k+1}}(X)} = \\
& \frac{1}{P_{n_k}(X)} \left(P_{n_{2k}}(X) \cdot \frac{P_{n_{2k}}(X^c)}{P_{n_{2k}}(X)} + P_{n_{2k+1}}(X) \cdot \frac{P_{n_{2k+1}}(X^c)}{P_{n_{2k+1}}(X)} \right) = \\
& \frac{1}{P_{n_k}(X)} (P_{n_{2k}}(X^c) + P_{n_{2k+1}}(X^c))
\end{aligned}$$

and by definition 4.4.3 (recall that $X^c := X^{\{c\}} = \{o \in X \mid (x, c) \in \mathcal{L}\}$)

$$\begin{aligned}
P_{n_{2k}}(X^c) + P_{n_{2k+1}}(X^c) &= \sum_{(o,c) \in \mathcal{L}} \mu_{n_{2k}}(o, m_k) + \sum_{(o,c) \in \mathcal{L}} \mu_{n_{2k+1}}(o, m_k) \\
&= \sum_{(o,c) \in \mathcal{L}} (\mu_{n_{2k}}(o, m_k) + \mu_{n_{2k+1}}(o, m_k))
\end{aligned}$$

applying proposition 4.4.4 we know

$$\sum_{(o,c) \in \mathcal{L}} (\mu_{n_{2k}}(o, m_k) + \mu_{n_{2k+1}}(o, m_k)) = \sum_{(o,c) \in \mathcal{L}} \mu_{n_k}(o, m_{P_k})$$

which is equal to $P_{n_k}(X^c)$ and thus,

$$\begin{aligned}
\rho_{m_k, n_{2k}} \rho_{m_k}(c|n_{2k}) + \rho_{m_k, n_{2k+1}} \rho_{m_k}(c|n_{2k+1}) &= \frac{1}{P_{n_k}(X)} \cdot P_{n_k}(X^c) \\
&= \rho_{m_{P_k}}(c|n_k).
\end{aligned}$$

□

Theorem 4.7.2: Suppose $J = 2$ and a impurity function ϕ (see definition 3.6.2) strictly concave, then $\text{range}(\nu) \subset [0, +\infty)$ i.e. the τ -impurity decreases for every “soft” splitting of the learning data set, i.e.,

$$\nu((q_k, s_k, m_k), n_k) = \Delta\tau(n_k, m_{P_k}) = \tau(n_k, m_{P_k}) - \rho_{m_k, n_{2k}} \tau(n_{2k}, m_k) - \rho_{m_k, n_{2k+1}} \tau(n_{2k+1}, m_k) \geq 0$$

Proof. By equation 4.14 of section 4.7.3 for a $J = 2$ classification problem

$$\rho_{m_k, n_{2k}} \tau(n_{2k}, m_k) + \rho_{m_k, n_{2k+1}} \tau(n_{2k+1}, m_k) = \rho_{m_k, n_{2k}} \phi(\rho_{m_k}(c|n_{2k})) + \rho_{m_k, n_{2k+1}} \phi(\rho_{m_k}(c|n_{2k+1}))$$

by proposition 4.7.3, we know that $\rho_{m_k, n_{2k}} + \rho_{m_k, n_{2k+1}} = 1$ and, thus, we can apply Jensen’s inequality for the strictly concave function ϕ and obtain

$$\rho_{m_k, n_{2k}} \phi(\rho_{m_k}(c|n_{2k})) + \rho_{m_k, n_{2k+1}} \phi(\rho_{m_k}(c|n_{2k+1})) \leq \phi(\rho_{m_k, n_{2k}} \rho_{m_k}(c|n_{2k}) + \rho_{m_k, n_{2k+1}} \rho_{m_k}(c|n_{2k+1})),$$

recall the theorem 4.7.1 and consequently

$$\phi\left(\rho_{m_k, n_{2k}} \rho_{m_k}(c|n_{2k}) + \rho_{m_k, n_{2k+1}} \rho_{m_k}(c|n_{2k+1})\right) = \phi\left(\rho_{m_{P_k}}(c|n_k)\right)$$

that, by definition, is the SODT impurity of node n_k , $\tau(n_k, m_{P_k})$, and, thus it is fulfilled that

$$\tau(n_k, m_{P_k}) - \rho_{m_k, n_{2k}} \tau(n_{2k}, m_k) - \rho_{m_k, n_{2k+1}} \tau(n_{2k+1}, m_k) \geq 0.$$

□

9

Graphical representation of the function ν of goodness

In figure 4.10 we represent an example of the crisp and SODT goodness functions. In this example the measurement space \mathcal{X} is one-dimensional, i.e. the number of attributes is $h = 1$. We can observe that in the crisp case (left part of the figure), the goodness function of all elements between two measurements is identical, giving as result a piecewise constant function which is non-differentiable. However, the SODT goodness of the elements is not constant but it is monotonously increasing or decreasing before and after each measurement, resulting in a smoother function which is differentiable w.r.t. the threshold value. This differentiability condition is explained in chapter 5.¹⁰

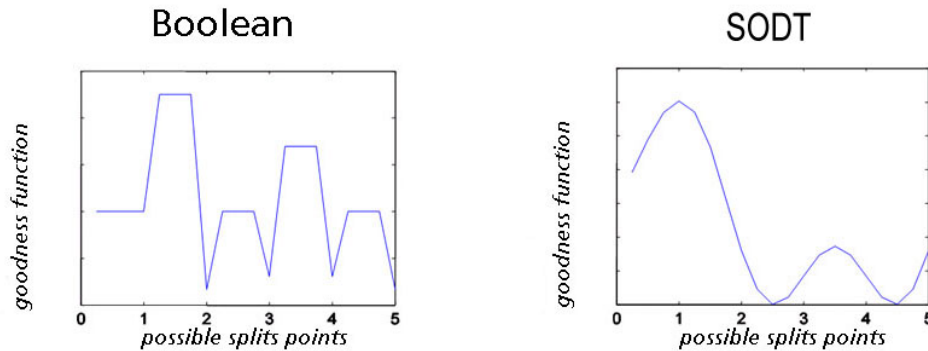


Figure 4.10: comparison crisp/Boolean and SODT goodness functions

4.7.5 Relationship between crisp DT and SODT

DT and SODT at the root node

Proposition 4.7.4: Let $T = (V, E)$ be a SODT and n_1 the root node, then for all $m \in (0, +\infty)$, the crisp Gini Index and the SODT Gini index coincide:

⁹due to the concavity of ν and the theorem of total probability

¹⁰In this figure, the goodness functions are analyzed at a discrete number of points.

$$i_{Gini}(n_1) = \tau_{Gini}(n_1, m)$$

Proof.

$$\tau(n_1, m) = \sum_{i \neq j} \rho_m(i|n_1) \rho_m(j|n_1)$$

for all $i, j \in \mathcal{C}$

$$\rho_m(j|n_1) = \frac{\sum_{(o,j) \in \mathcal{L}} \mu_{n_1}(o, m)}{\sum_{o \in X} \mu_{n_1}(o, m)} = \frac{\sum_{(o,j) \in \mathcal{L}} 1}{\sum_{o \in X} 1} = \frac{N_j(n_1)}{N(n_1)} = p(j|n_1)$$

because it is assumed that the MSD of any $o \in \mathcal{X}$ to the root node is equal to 1. \square

Asymptotical behaviour of the node impurity function τ

Lemma 4.7.2: Let $FL : \mathbb{R}^2 \times (0, +\infty) \rightarrow [0, 1]$ be the Fuzzy Less operator introduced in definition 4.3.1. Then, for $m \rightarrow \infty$, the operator FL tends to the boolean less operator, i.e.,

$$\begin{aligned} \lim_{m \rightarrow \infty} FL(x, x + \Delta x, m) &= 1 \quad \forall \Delta x > 0 \\ \lim_{m \rightarrow \infty} FL(x, x - \Delta x, m) &= 0 \quad \forall \Delta x > 0 \end{aligned}$$

where $x \in \mathbb{R}$.

Proof.

$$\lim_{m \rightarrow \infty} FL(x, x + \Delta x, m) = \lim_{m \rightarrow \infty} \frac{\tanh((x + \Delta x - x) \cdot m) + 1}{2} = \frac{1 + 1}{2} = 1$$

Analogously

$$\lim_{m \rightarrow \infty} FL(x, x - \Delta x, m) = \lim_{m \rightarrow \infty} \frac{\tanh(-\Delta x \cdot m) + 1}{2} = 0$$

\square

Lemma 4.7.3: The degree of membership of an element $o \in \mathcal{X}$ to a node $n_i, i = 1, \dots, L$ of a tree T tends to the boolean indicator function $I : \mathcal{X} \rightarrow \{0, 1\}$ as m tends to infinity.

$$\forall i = 1, \dots, L \quad \lim_{m \rightarrow \infty} \mu_{n_i}(o, m) = \begin{cases} 1 & \text{if } o \in n_i \\ 0 & \text{else} \end{cases}$$

Proof. We will do this proof by induction

$i = 1 :$

$$\lim_{m \rightarrow \infty} \mu_{n_1}(o, m) = \lim_{m \rightarrow \infty} 1 = 1$$

$\forall i' < i \Rightarrow i :$

$$\mu_{n_i}(o, m) = \begin{cases} FL(o, \sigma, m) \cdot \mu_{n_{\frac{i}{2}}}(o, m) & \text{if } i \equiv 0 \pmod{2} \\ FG(o, \sigma, m) \cdot \mu_{n_{\frac{i-1}{2}}}(o, m) & \text{if } i \equiv 1 \pmod{2} \end{cases}$$

Case 1: $i \equiv 0 \pmod{2}$

$$\lim_{m \rightarrow \infty} \mu_{n_i}(o, m) = \lim_{m \rightarrow \infty} (FL(o, \sigma, m) \cdot \mu_{\frac{i}{2}}(o, m))$$

$$\text{Lemma 4.7.2} \begin{cases} \lim_{m \rightarrow \infty} \mu_{\frac{i}{2}}(o, m) & \text{if } o < \sigma \\ 0 & \text{else} \end{cases} = \begin{cases} 1 & \text{if } (o < \sigma_{q_{\frac{i-1}{2}}}) \wedge (o < \sigma) \\ 0 & \text{else} \end{cases}$$

$$(o \in n_i \Rightarrow o \in n_{\frac{i}{2}} \text{ if } i \text{ even})$$

Case 2: $i \equiv 1 \pmod{2}$

$$\lim_{m \rightarrow \infty} \mu_{n_i}(o, m) = \lim_{m \rightarrow \infty} (FG(o, \sigma, m) \cdot \mu_{\frac{i-1}{2}}(o, m))$$

$$\text{Lemma 4.7.2} \begin{cases} \lim_{m \rightarrow \infty} \mu_{\frac{i-1}{2}}(o, m) & \text{if } o > \sigma \\ 0 & \text{else} \end{cases} = \begin{cases} 1 & \text{if } (o > \sigma_{q_{\frac{i}{2}}}) \wedge (o > \sigma) \\ 0 & \text{else} \end{cases}$$

$$(o \in n_i \Rightarrow o \in n_{\frac{i-1}{2}} \text{ if } i \text{ odd})$$

□

Proposition 4.7.5: Let T be a SODT and let $n \in V(T)$. Let us suppose $\rho_m(j|n)$ as expounded in definition 4.7.2. Then,

$$\lim_{m \rightarrow \infty} \rho_m(j|n) = p(j|n)$$

Proof.

$$\lim_{m \rightarrow \infty} \rho_m(j|n) = \lim_{m \rightarrow \infty} \frac{P_n(X^j)}{P_n(X)} = \lim_{m \rightarrow \infty} \frac{\sum_{(o,j) \in \mathcal{L}} \mu_n(o, m)}{\sum_{o \in X} \mu_n(o, m)} = \frac{N_j(n)}{N(n)} = p(j|t)$$

□

Theorem 4.7.3: Let us suppose the conditions of proposition 4.7.5 to be fulfilled and let $\tau : \text{Nodes}(T) \times (0, +\infty) \rightarrow \mathbb{R}$ be the SODT impurity function exposed in definition 4.7.5, then

$$\lim_{m \rightarrow \infty} \tau(n, m) = i(n)$$

Proof. Trivial by application of proposition 4.7.5

□

We conclude then that when the unpreciseness factors m_i at each node $n_i \in V \setminus V_{\text{leaves}}$ tend to infinity, our SODT tends to a crisp decision tree.

4.7.6 Variable combinations: Oblique SODT

The SODT described up to now uses a standard structure, where the test at each internal node is done in one of the variables, i.e., all splits are perpendicular to the coordinate axes. There are some situations where the class structure depends on combination of variables, for instance, the data illustrated in figure 4.11. The SODT would need to split many times to approximate the separating hyperplane by parallels to the axes. In such problems, the set of allowable splits is extended so that it includes all possible combination splits of the form

$$\text{Is } h(x) \leq \sigma ?$$

where h is a function $h : \mathcal{X} \rightarrow \mathbb{R}$. In the concrete case that a linear structure is suspected, then the set of potential splits would include all linear combination splits of the form

$$\text{Is } h(x) = \sum_m a_m x_m \leq \sigma ?$$

and the resulting decision tree is called **oblique** decision tree. An algorithm was developed by CART ([10]) to search through such splits in an effort to find the one that maximizes the goodness of split criterion.

In the case of SODT, we would proceed in the same way as before but redefining the functions FL and FG as

$$FL(o, \sigma, m) := \frac{\tanh((\sigma - h(o)) \cdot m) + 1}{2}$$

and

$$FG(o, \sigma, m) := \frac{\tanh((h(o) - \sigma) \cdot m) + 1}{2}$$

Analogously as in the crisp case, if $h(x)$ is a linear function, the resulting SODT is called **oblique** SODT.

This is another significant advantage of SODT, that due to the fuzzification in the operators, the nodes can easily be described just by changing once the operator, without needing to redefine all fuzzy sets.

4.8 SODT Stopping Rule

In section 3.12 we gave an overview of the stopping rules for the crisp case. In order to know when to stop splitting a SODT, this rules need to be modified as follows:

Let μ , τ and ν be the functions defined in sections 4.4, 4.7.3 and 4.7.4, respectively, then

- One option is to set a threshold $\beta > 0$ and declare a node n terminal if

$$\max_{s \in S} \nu(s, n, m) < \beta$$

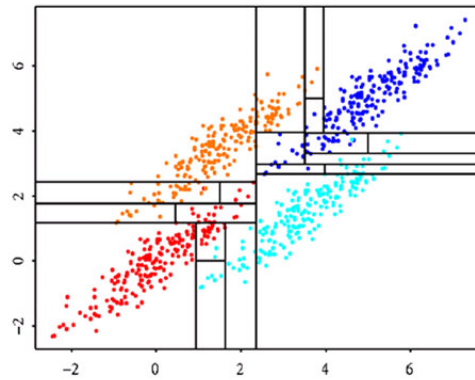


Figure 4.11: Example of a data set where splitting with variable combinations would fit better

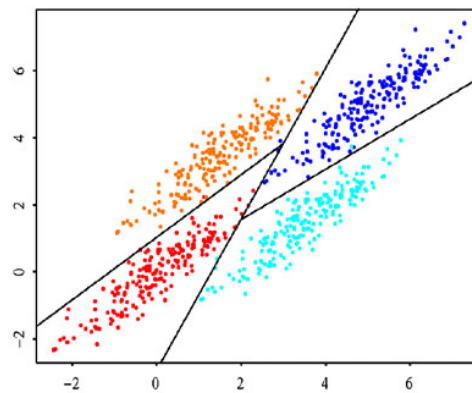


Figure 4.12: Example of variable combinations for splitting

where \mathcal{S} represents the set of all allowable splits on n and m the unpreciseness factor associated to its parent node, i.e. if

$$\max_{s \in \mathcal{S}} \Delta\tau(s, n, m) < \beta$$

- Threshold on the SODT node size: A node $n \in V$ is declared terminal if the sum of the degrees of membership to node n of the elements of the learning set is smaller than a certain $k \in \mathbb{N}$. In other words, if $\sum_{x \in \mathcal{X}} \mu_n(x) < k$, then stop splitting and consider $n \in V_{leaves}$.
- Two-stage SODT search: As well as in the crisp approach, this approach is also divided in two subtasks:
 - Determine the structure of the tree
 - Find splits for all the nodes, i.e., in the SODT case, find the associated 3-tuple (q, σ, m) for each node $n \in V$.

4.9 SODT terminal node class assignment

The class assignment process in a SODT is very similar to the class assignment process in a crisp decision tree (see section 3.13).

The class label assigned to a SODT leaf node n is the class with the highest probability in this node n , i.e., the function

$$l : V_{leaves} \longrightarrow \mathcal{C}$$

that assigns a class label to each SODT terminal node is then

$$l(n) = \arg \max_{j \in \mathcal{C}} \rho_m(j|n).$$

The function $\rho_m : \mathcal{C} \longrightarrow [0, 1]$ is defined in section 4.7.1 as

$$\rho_m(j|n) = \frac{P_n(X^j)}{P_n(X)} \quad \forall j \in J,$$

Thus, we can write

$$l(n) = \arg \max_{j \in \mathcal{C}} \rho_m(j|n) = \arg \max_{j \in \mathcal{C}} P_n(X^j) = \arg \max_{j \in \mathcal{C}} \sum_{\substack{(x^{(i)}, y^{(i)}) \in \mathcal{L} \\ y^{(i)} = j}} \mu_n(x^{(i)}).$$

Recall that in a crisp decision tree, the class assignment rule is

$$l(n) = \arg \max_{j \in J} p(j|n),$$

which can be written as

$$l(n) = \arg \max_{j \in J} \left\{ \# \left\{ i \mid x^{(i)} \in R_n \wedge y^{(i)} = j \right\} \right\}.$$

4.10 Algorithm for the learning process of a SODT

We start with an empty SODT and the complete learning set $\mathcal{L} \subseteq \mathcal{X} \times \mathcal{C}$ and then proceed in the following way:

- If $\exists c \in \mathcal{C}$ such that for all objects $(x, y) \in \mathcal{L}$ it is fulfilled that $y = c$, then create a terminal node with label class c and stop.
- Otherwise, score each one of the set of possible split points $\mathcal{S} \subseteq \{1, \dots, h\} \times \mathcal{X} \times [0, +\infty)$ using a SODT goodness measure (see section 4.7.4), i.e., choose q^*, σ^*, m^* so that

$$\nu(q^*, \sigma^*, m^*, n) = \max_{\sigma, q, m} \nu(q, \sigma, m, n)$$

- Choose the best split point $s^* = (q^*, \sigma^*, m^*)$ as the test and create the two corresponding child nodes for $x_{q^*} <^{FL} \sigma^*$ and $x_{q^*} >^{FG} \sigma^*$, respectively.
- Label the node as (q^*, σ^*, m^*) and proceed in the same way with the child nodes until the chosen stopping criteria is fulfilled.
- Assign class labels to each terminal node as explained in section 4.9

Note: The domain of definition of the unpreciseness factor m is $(0, +\infty)$, but we can also define a discrete domain of definition where the values of m are taken as a function of the learning sample. In table 4.10 we can see how this values can be taken depending on the level of preciseness of m which is required (see [43] or [42]).

Soft comparison operator	FL, FG
very unprecise	$m = \frac{\operatorname{artanh}(\sqrt{0.25})}{0.25 \cdot (x_{max} - x_{min})}$
unprecise	$m = \frac{\operatorname{artanh}(\sqrt{0.30})}{0.20 \cdot (x_{max} - x_{min})}$
mean	$m = \frac{\operatorname{artanh}(\sqrt{0.35})}{0.15 \cdot (x_{max} - x_{min})}$
precise	$m = \frac{\operatorname{artanh}(\sqrt{0.40})}{0.10 \cdot (x_{max} - x_{min})}$
very precise	$m = \frac{\operatorname{artanh}(\sqrt{0.45})}{0.05 \cdot (x_{max} - x_{min})}$

Figure 4.13: determining the “unpreciseness” of the soft comparison operators

4.11 SODT inference or classification

Recall that:

- Discrimination is the process of deriving classification rules from a given sample of classified objects $\mathcal{L} \subseteq \mathcal{X} \times \mathcal{C}$. (Learning)
- Classification refers to the application of the rules to new objects of unknown class, i.e., classification consists of applying the rules to determine the class of new objects of unknown class. (Tree inference)

Once the SODT is already created based on a set of classified data \mathcal{L} and, eventually, also on previously given information, the next step is to assign a class to new objects of unknown class. In a crisp decision tree, this class is uniquely determined for every measurement vector by passing it through the tree starting at the root node and arriving at a leaf node. The class assigned to the measurement vector will be the class label assigned to the leaf at which it arrives. In a SODT, however, the measurement vector, after going through the tree, will not arrive just at one of the terminal nodes, but at several ones with degrees of membership that will sum up to one. Depending on what is the interest of the classification, there are two possibilities to follow. The first possibility is to classify this measurement vector, not just to one class, but to the class labels of the leaf nodes it reaches with its degrees of membership. On the other hand, in case a unique output class is needed, the second possibility is to make a compromise with the degrees of membership to the different leaf nodes and compute finally only one class as output of the measurement vector, i.e. we can defuzzify the output obtained to give a crisp result or to give a result with degrees of membership to the leaf nodes, and, thus, to the classes.

- First option: SODT as a Fuzzy classifier (see definition 4.2.1).

An element $x \in \mathcal{X}$ belongs to all leaf nodes with certain degrees of membership. Each leaf node $n_k \in V_{leaves}(T)$ has a class label $l(n_k)$ associated (see section 4.9) and the classification of the measurement vector x is given as a J -dimensional vector where each component $j \in \{1, \dots, J\}$ is the membership degree to class $j \in \mathcal{C}$ i.e., we give a non-defuzzified class label as follows:

$$d(x) = \left(\mu^1(x), \mu^2(x), \dots, \mu^J(x) \right) \in [0, 1]^J$$

where $\mu^j(x)$ is the MSD (membership degree) to each class $j \in \mathcal{C} = \{1, \dots, J\}$

$$\mu^j(x) = \sum_{\substack{n_k \in V_{leaves}(T) \\ l(n_k)=j}} \mu_k(x).$$

To calculate the degree of membership $\mu_k(x)$ of x to a node n_k , see definition 4.4.1.

Claim: The sum of the membership degrees to all classes $j \in \mathcal{C}$ of an element $x \in \mathcal{X}$ is one, i.e.,

$$\sum_{j=1}^J \mu^j(x) = 1.$$

Proof. We have

$$\begin{aligned} \sum_{j=1}^J \mu^j(x) &= \sum_{j=1}^J \sum_{\substack{n_k \in V_{leaves}(T) \\ l(n_k)=j}} \mu_k(x) \\ &= \sum_{n_k \in V_{leaves}(T)} \mu_k(x) \end{aligned}$$

since for all $n_k \in V_{leaves}(T)$ there exists one and only one $j \in \mathcal{C} = \{1, \dots, J\}$ such that $l(n_k) = j$ (see section 4.9). Besides, if we apply lemma 4.4.4, then

$$\sum_{n_k \in V_{leaves}(T)} \mu_k(x) = 1 \quad \forall x \in \mathcal{X}.$$

Therefore, we can conclude that

$$\sum_{j=1}^J \mu^j(x) = 1.$$

□

- Second option: SODT as a (crisp) classifier (see definition 3.3.5).

The classification of $x \in \mathcal{X}$ is the class label of the leaf node to which x belongs with the highest membership degree, then

$$d(x) = l\left(\arg \max_{n_k \in V_{leaves}(T)} \mu_k(x)\right) \in \mathcal{C} = \{1, \dots, J\},$$

where l is the SODT terminal node class assignment explained in section 4.9,

4.12 SODT Misclassification Rate

The true misclassification rate, as we explained in section 3.14, is often difficult to compute, since in most of the cases the data in \mathcal{L} are available with no possibility of knowing the distribution function they follow. Thus, the true misclassification rate is estimated and \mathcal{L} must be applied for this estimation, as well as for the construction of the classifier. The estimation of the true misclassification rate of a SODT can, analogously to the case of any classifier, be done in several ways. For each estimation, we distinguish between the two cases of SODT explained in section 4.11:

- SODT as a classifier
- SODT as a fuzzy classifier

SODT Resubstitution estimate

Recall that in the crisp case, the “resubstitution estimate” is the most commonly used estimate for the true misclassification rate.

Definition 4.12.1: Let $\mathcal{L} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ be a learning sample, where $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{C}$. Let a SODT classifier be constructed from \mathcal{L} . We define two types of SODT resubstitution estimates, depending on whether the final classification of an object is given with degrees of membership or defuzzified (i.e., depending on whether it is a fuzzy classifier or a classifier in the usual way). In the case that we consider a SODT as an usual classifier $d : \mathcal{X} \rightarrow \mathcal{C}$, the SODT resubstitution estimate, denoted by $\mathcal{R}(d)$, is given like in the crisp case, by

$$\mathcal{R}(d) = \frac{1}{N} \sum_{i=1}^N I(d(x^{(i)}) \neq y^{(i)})$$

where $I(\cdot)$ is the indicator function, being equal to one if the event inside the parentheses is true and otherwise being zero. Which means that if we consider the option that the classification of $(x^{(i)})$ is the class label of the leaf node to which $(x^{(i)})$ belongs with the highest membership degree, i.e., if

$$d(x^{(i)}) = l\left(\arg \max_{n_k \in V_{leaves}(T)} \mu_k(x^{(i)})\right) \in \mathcal{C} = \{1, \dots, J\},$$

where l is the SODT terminal node class assignation explained in section 4.9, then

$$\mathcal{R}(d) = \frac{1}{N} \sum_{i=1}^N I\left(l\left(\arg \max_{n_k \in V_{leaves}(T)} \mu_k(x^{(i)})\right) \neq y^{(i)}\right).$$

In the case we consider a SODT as a fuzzy classifier $d : \mathcal{X} \rightarrow \{(p_1, \dots, p_J) \mid p_i \in [0, 1]\}$, where an element (object) belongs to all leaf nodes with certain degrees of membership, i.e., if we give a non-defuzzified class label as follows:

$$d(x^{(i)}) = \left(\mu^1(x^{(i)}), \mu^2(x^{(i)}), \dots, \mu^J(x^{(i)})\right) \in [0, 1]^J$$

where $\mu^j(x^{(i)})$ is the degree of membership to each class $j \in \mathcal{C} = \{1, \dots, J\}$

$$\mu^j(x^{(i)}) = \sum_{\substack{n_k \in V_{leaves}(T) \\ l(n_k)=j}} \mu_k(x^{(i)})$$

then, the SODT resubstitution estimate is of the form:

$$\mathcal{R}(d) = \frac{\sum_{i=1}^N \sum_{\substack{n_k \in V_{leaves}(T) \\ l(n_k) \neq y^{(i)}}} \mu_k(x^{(i)})}{\sum_{i=1}^N \sum_{n_k \in V_{leaves}(T)} \mu_k(x^{(i)})}$$

where $\sum_{i=1}^N \sum_{n_k \in V_{leaves}(T)} \mu_k(x^{(i)})$ is equal to N . Thus, the resulting SODT Resubstitution estimate is

$$\mathcal{R}(d) = \frac{1}{N} \sum_{i=1}^N \sum_{\substack{n_k \in V_{leaves}(T) \\ l(n_k) \neq y^{(i)}}} \mu_k(x^{(i)}).$$

Remark: The test sample estimate and the V-fold-cross-validation for SODT are computed analogously. We incorporate the formulas given above for the SODT resubstitution estimate in the definitions of test sample estimate and the V-fold-cross-validation given in section 3.14.

4.13 SODT for a fuzzified data sample

Following, we will see that a SODT can also be created and applied in the case that the data are given as fuzzified data samples, i.e., in a linguistic form with uncertainty, where the output of each element of the learning sample indicates the probability of an object to belong to one of the classes fixed in advance (see [36]). Normally, just two classes are considered in order to have exactly one output for each instance, which would be the degree of membership to the fixed class. The degree of membership to the other class is then trivially calculated.

Definition 4.13.1: Let us consider a finite learning sample $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times [0, 1]$, where, for all $o \in X$, we call $\tilde{\mu}_c(o) \in Y \subseteq [0, 1]$ the degree of membership of an element $x \in \mathcal{X}$ to a class $c \in \mathcal{C} = \{c, c'\}$ previously chosen. This sample is called a fuzzified data sample. The degree of membership of this element to the other class is then $\tilde{\mu}_{c'}(x) = 1 - \tilde{\mu}_c(x)$.

Definition 4.13.2: Let $(\mathcal{C}, \mathcal{F}_{\mathcal{C}})$ be the measurable space composed of the set of classes $\mathcal{C} = \{c, c'\}$ and the σ -algebra $\mathcal{F}_{\mathcal{C}}$, which is the power set of \mathcal{C} (see section 4.7.1). Let $\mathcal{L} = (X, Y) \subseteq \mathcal{X} \times [0, 1]$ be a fuzzified data sample and T a SODT, then for all $n \in V(T)$, we define a set function $\tilde{\Psi} : \mathcal{F}_{\mathcal{C}} \rightarrow [0, 1]$ such that for all $C \in \mathcal{F}_{\mathcal{C}} = \{\emptyset, \mathcal{C}, \{c\}, \{c'\}\}$

$$\tilde{\Psi}(C) = \frac{\sum_{\substack{o \in X \\ c \in C}} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)}.$$

Lemma 4.13.1: Let us suppose that the same conditions of the previous definition are fulfilled, then the set function $\tilde{\Psi} : \mathcal{F}_{\mathcal{C}} \rightarrow [0, 1]$ is a measure and thus $(\mathcal{C}, \mathcal{F}_{\mathcal{C}}, \tilde{\Psi})$ is a **measure space**.

Proof. We have

- $0 \leq \tilde{\Psi}(C)$ for any $C \in \mathcal{F}_{\mathcal{C}}$ since

$$\tilde{\Psi}(C) = \frac{\sum_{\substack{o \in X \\ c \in C}} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)}$$

and for all $n \in V(T)$, $o \in X$, it is fulfilled that $\mu_n(o) \in [0, 1]$ (see proposition 4.4.1) and $\tilde{\mu}_c(o) \in [0, 1]$ by definition.

- $\tilde{\Psi}(\emptyset) = 0$ since

$$\tilde{\Psi}(\emptyset) = \frac{\sum_{\substack{o \in X \\ c \in \emptyset}} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)}.$$

- If $C_i \in \mathcal{F}_C$, $i = 1, 2, \dots$ and C_i 's are mutually disjoint, i.e., $C_i \cap C_k = \emptyset$ for any $i \neq k$, then $\tilde{\Psi}\left(\bigcup_{j=1}^{\infty} C_i\right) = \sum_{j=1}^{\infty} \tilde{\Psi}(C_i)$ since the unique non-trivial disjoint sets in \mathcal{F}_C are $C = \{c\}$ and $C' = \{c'\}$ and

$$\begin{aligned} \tilde{\Psi}(C \cup C') &= \frac{\sum_{\substack{o \in X \\ c \in C \cup C'}} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)} \\ &= \frac{\sum_{\substack{o \in X \\ c \in C}} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)} + \frac{\sum_{\substack{o \in X \\ c \in C'}} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)} \\ &= \tilde{\Psi}(C) + \tilde{\Psi}(C') \end{aligned}$$

□

Proposition 4.13.1: Let us suppose that the same conditions of definition 4.13.2 are fulfilled. Then, the set function $\tilde{\Psi} : \mathcal{F}_C \longrightarrow [0, 1]$ is a **probability measure** and thus $(\mathcal{C}, \mathcal{F}_C, \tilde{\Psi})$ is a **probability space**.

Proof. By the previous lemma, $(\mathcal{C}, \mathcal{F}_C, \tilde{\Psi})$ is a measure space. Then for proving that it is also a probability space we just need to show the additional condition that $\tilde{\Psi}(\mathcal{C}) = 1$:

$$\begin{aligned} \tilde{\Psi}(\mathcal{C}) &= \frac{\sum_{\substack{o \in X \\ j \in \mathcal{C}}} \mu_n(o) \tilde{\mu}_j(o)}{\sum_{o \in X} \mu_n(o)} \\ &\stackrel{C=\{c, c'\}}{=} \frac{\sum_{o \in X} \mu_n(o) \tilde{\mu}_c(o)}{\sum_{o \in X} \mu_n(o)} + \frac{\sum_{o \in X} \mu_n(o) \tilde{\mu}_{c'}(o)}{\sum_{o \in X} \mu_n(o)} \\ &= \frac{\sum_{o \in X} \mu_n(o) (\tilde{\mu}_c(o) + \tilde{\mu}_{c'}(o))}{\sum_{o \in X} \mu_n(o)} \\ &\stackrel{\substack{\text{By def. 4.13.1} \\ \tilde{\mu}_c(o) + \tilde{\mu}_{c'}(o) = 1}}{=} \frac{\sum_{o \in X} \mu_n(o)}{\sum_{o \in X} \mu_n(o)} = 1 \end{aligned}$$

□

Definition 4.13.3: Let $(\mathcal{C}, \mathcal{F}_C, \tilde{\Psi})$ be the probability space defined in proposition 4.13.1 for a node $n \in V(T)$. Let \mathcal{L} be a fuzzy data sample and T a SODT. For all $n \in V(T)$, we

define the fuzzy probability of a class $j \in \mathcal{C} = \{c, c'\}$ in a SODT node n as the function $\tilde{\rho} : [0, 1] \longrightarrow [0, 1]$

$$\tilde{\rho}(j|n) = \tilde{\Psi}(\{j\}) \quad \forall j \in \mathcal{C}$$

where $\tilde{\Psi} : \mathcal{F}_{\mathcal{C}} \longrightarrow [0, 1]$ is the probability function defined above for each fixed node $n \in V(T)$.

Note: The function $\tilde{\rho} : [0, 1] \longrightarrow [0, 1]$ would substitute the function $\rho : \mathcal{C} \longrightarrow [0, 1]$ (see definition 4.7.2) during the whole learning and inference process, in case the learning sample is a fuzzy data sample.

4.14 Distance between SODTs

In this section we define a distance between two SODTs. This distance is analogue to the distance between two (crisp) decision trees that we proposed in section 3.17. Thus, it establish a compromise between the differences of the trees from the graph theory point of view (see chapter 2), and the differences between the associated 3-tuplas to each of their nodes (see chapter 4).

Notation We will consider a SODT $T = (V, E)$ (see definition 4.3) with the enumeration of its nodes that we proposed in theorem 2.7.3, i.e.

$$E = \{(n_i, n_j) \in (V \setminus V_{leaves}) \times V \mid j = 2i \vee j = 2i + 1\}.$$

Let $\hat{\Omega}$ be the space containing all possible SODTs defined in this way and \mathcal{D} the space of all possible learning samples.

Let $T_j = (V_j, E_j)$, $j = 1, 2$ be a SODT where each node $i \in V_j$, $j = 1, 2$ has a 3-tuple $n_i^{T_j} = (q_i^{T_j}, \sigma_i^{T_j}, m_i^{T_j})$ associated. Let O_{T_j} be the order of the tree (see definition 2.2.1 in chapter 2).

Our aim in this section is to define a distance which measures the difference in the structure between two SODTs. Analogously to the definition of distance between two (crisp) decision trees that we proposed in section 3.17, we define the distance between SODTs as the sum of the distances between the nodes of the trees. In this case, more concretely, between the 3-tuplas associated to each node. Thus, we are comparing the 3-tuples $n_i^{T_1} = (q_i^{T_1}, \sigma_i^{T_1}, m_i^{T_1})$ and $n_i^{T_2} = (q_i^{T_2}, \sigma_i^{T_2}, m_i^{T_2})$ associated to node $i \in V_j$, $j = 1, 2$. Since two trees do not necessarily have the same order, then we complete trees T_1 and T_2 in the same way we explained in section 3.17, i.e., such that $O(T_1) = O(T_2) = i^*$, where

$$i^* = \max\{i \mid n_i \in T_1 \vee n_i \in T_2\},$$

and assign a 3-tuple (Q, Σ, M) to the nodes which are in T_1 but not in T_2 or vice versa. The values assigned to this 3-tuple are penalizing, in the sense that the distance between

a node and this node with an assigned 3-tupla and will be equal to the maximal possible distance between two nodes.

Thus, without loss of generalization, we can suppose that the two trees have the same order.

Definition 4.14.1: Let $T = (V, E)$ and $T' = (V', E')$ be two SODTs of the same order O where each node of T and T' has a 3-tuple $n_i = (q_i, \sigma_i, m_i) \in \Theta \times \mathbb{R} \times (0, +\infty)$ and $n'_i = (q'_i, \sigma'_i, m'_i) \in \Theta \times \mathbb{R} \times (0, +\infty)$ associated, respectively. We define the η -distance between n_i and n'_i as a function

$$d_\eta : (\Theta \times \mathbb{R} \times (0, +\infty)) \times (\Theta \times \mathbb{R} \times (0, +\infty)) \rightarrow [0, +\infty)$$

$$\begin{aligned} d_\eta(n_i, n'_i) &= d_\eta((q_i, \sigma_i, m_i), (q'_i, \sigma'_i, m'_i)) \\ &:= \min(\sqrt{(d_v(q_i, q'_i))^2 + (d_s(n_i, n'_i))^2 + (d_m(n_i, n'_i))^2}, 1) \end{aligned}$$

where $d_v : \Theta \times \Theta \rightarrow [0, +\infty)$ is defined as we proposed in definition 3.17.1 for a (crip) decision tree:

$$d_v(q_i, q'_i) := \begin{cases} 0 & \text{if } q_i = q'_i \\ 1 & \text{else} \end{cases}$$

i.e., as the *discrete metric*, and $d_s : (\Theta \times \mathbb{R} \times (0, +\infty)) \times (\Theta \times \mathbb{R} \times (0, +\infty)) \rightarrow [0, +\infty)$ is defined as

$$d_s(n_i, n'_i) := \frac{|\sigma_i - \sigma'_i|}{1 + |\sigma_i - \sigma'_i|} \cdot (1 - d_v(q_i, q'_i))$$

and $d_m : (\Theta \times \mathbb{R} \times (0, +\infty)) \times (\Theta \times \mathbb{R} \times (0, +\infty)) \rightarrow [0, +\infty)$ is defined as

$$d_m(n_i, n'_i) := \frac{|m_i - m'_i|}{1 + |m_i - m'_i|} \cdot (1 - d_v(q_i, q'_i)).$$

Lemma 4.14.1: Let $n = (q, \sigma, m)$, $n' = (q', \sigma', m') \in \Theta \times \mathbb{R} \times (0, +\infty)$. If we call

$$\tilde{d}_m(m, m') = \frac{|m - m'|}{1 + |m - m'|},$$

then $\tilde{d}_m : \mathbb{R} \times \mathbb{R} \rightarrow [0, 1]$ is a metric on \mathbb{R} . Moreover, for d_s , d_v and d_m it is fulfilled that

$$d_s(n, n') \in [0, 1], \quad d_m(n, n') \in [0, 1]$$

and that if $q \neq q'$, then $d_v(q, q') = 1$, $d_s(n, n') = 0$ and $d_m(n, n') = 0$.

Proof. Analogously to the proof of lemma 3.17.1.

Note: Recall that in lemma 3.17.1 we proved that if we define:

$$\tilde{d}_s(\sigma, \sigma') = \frac{|\sigma - \sigma'|}{1 + |\sigma - \sigma'|},$$

then $\tilde{d}_s : \mathbb{R} \times \mathbb{R} \longrightarrow [0, 1]$ is a metric on \mathbb{R} . □

Lemma 4.14.2: Let $a = (q, \sigma, m)$, $b = (q', \sigma', m')$, $c = (q'', \sigma'', m'') \in \Theta \times \mathbb{R} \times (0, +\infty)$ such that $q = q'$ and $q \neq q''$ then:

a) $0 \leq d_\eta(a, b) \leq 1$

b) $d_\eta(a, c) = 1$

and thus $d_\eta(a, b) \leq d_\eta(a, c)$, which in a SODT context means that, if the splitting variables are different, then the distance is bigger than in the case they are the same and the threshold or/and the unpreciseness factor are different.

Proof. a) Trivial by definition of $d_\eta(a, b) \leq 1$.

b) It is also trivial that $d_\eta(a, c) = 1$ since

$$\begin{aligned} d_\eta(a, c) &= \sqrt{(d_v(q, q''))^2 + (d_s(n, n''))^2 + (d_m(n, n'))^2} \\ &\stackrel{\text{Lemma 4.14.1}}{=} \sqrt{(d_v(q, q''))^2} = 1 \end{aligned}$$

□

Lemma 4.14.3: Let $D : X \times X \longrightarrow \mathbb{R}$ be a metric/distance function on a set X , then the function $d : X \times X \longrightarrow \mathbb{R}$ defined as

$$d(x, y) = \min(D(x, y), 1),$$

is a metric on X .

Proof. It can be easily proved that d fulfills the axioms of a metric:

a) $d(x, y) \geq 0$ ($= 0 \Leftrightarrow x = y$) (Non negativity) (identity of indiscernibles)

b) $d(x, z) \leq d(x, y) + d(y, z)$ (Triangular inequality)

We distinguish between two possible cases:

$$\begin{aligned} \text{Case 1: } D(x, y) \leq 1 \wedge D(y, z) \leq 1 &\Rightarrow d(x, y) = D(x, y) \wedge d(y, z) = D(y, z) \\ &\Rightarrow d(x, z) \leq D(x, z) \leq d(x, y) + d(y, z). \end{aligned}$$

$$\begin{aligned} \text{Case 2: } D(x, y) \geq 1 \vee D(y, z) \geq 1 &\Rightarrow d(x, y) = 1 \vee d(y, z) = 1 \\ &\Rightarrow d(x, z) \leq 1 \leq d(x, y) + d(y, z). \end{aligned}$$

$$c) d(x, y) = d(y, x) \quad (\text{Symmetry})$$

□

Proposition 4.14.1: Let $d_\eta : (\Theta \times \mathbb{R} \times (0, +\infty)) \times (\Theta \times \mathbb{R} \times (0, +\infty)) \rightarrow [0, +\infty)$ be the function depicted in definition 4.14.1 and $\Theta = \{1, 2, \dots, h\}$, where h is the number of variables of the measurement space, then d_η is a metric on $\Theta \times \mathbb{R} \times (0, +\infty)$. Thus, $(\Theta \times \mathbb{R} \times (0, +\infty), d_\eta)$ is a metric space.

Proof. Let us see that d_η satisfies the requirements of a metric:

- a) $d_\eta(n, n') \geq 0$ ($= 0 \Leftrightarrow n = n'$) (Non negativity) (identity of indiscernibles)
- b) $d_\eta(n, n'') \leq d_\eta(n, n') + d_\eta(n', n'')$ (Triangular inequality)
- c) $d_\eta(n, n') = d_\eta(n', n)$ (Symmetry)

a) and c) are trivially fulfilled.

b) Let us see that

$$d_\eta(\overbrace{(q, \sigma, m)}^n, \overbrace{(q', \sigma', m')}^{n'}) + d_\eta(\overbrace{(q', \sigma', m')}^{n'}, \overbrace{(q'', \sigma'', m'')}^{n''}) \geq d_\eta(\overbrace{(q, \sigma, m)}^n, \overbrace{(q'', \sigma'', m'')}^{n''})$$

Case 1: $q = q'$ then, by lemma 4.14.2, $d_\eta(n, n') \in [0, 1]$

$$\text{Case 1.1: } q = q' = q'' \Rightarrow d_\eta(n, n') = \min \left(\sqrt{(\tilde{d}_s(\sigma, \sigma'))^2 + (\tilde{d}_m(m, m'))^2}, 1 \right)$$

and analogously with $d_\eta(n', n'')$ and $d_\eta(n, n'')$.

Since \tilde{d}_s and \tilde{d}_m are metrics on \mathbb{R} (see lemmas 3.17.1 and 4.14.1),

then, in this case, $\sqrt{(\tilde{d}_s(\sigma, \sigma'))^2 + (\tilde{d}_m(m, m'))^2}$ is a metric based

on the Pythagorean theorem analogously to the Euclidean

distance. Thus, by lemma 4.14.3, d_η is a metric and consequently,

the triangular inequality is fulfilled.

$$\text{Case 1.2: } q = q' \wedge q \neq q'' \Rightarrow d_\eta(n, n') + d_\eta(n', n'') > 1 = d_\eta(n, n'')$$

since $d_\eta(n, n') \in [0, 1]$ and $d_\eta(n', n'') = 1$

Case 2: $q \neq q'$ then, by lemma 3.17.2, $d_\eta(n, n') = 1$

Case 2.1: $q = q'' \wedge q' \neq q'' \Rightarrow d_\eta(n, n') + d_\eta(n', n'') = 1 + 1 >$
 $> d_\eta(n, n'') \in [0, 1]$

Case 2.2: $q \neq q'' \wedge q' \neq q'' \Rightarrow d_\eta(n, n') + d_\eta(n', n'') = 1 + 1 > d_\eta(n, n'') = 1$

Case 2.3: $q \neq q'' \wedge q' = q'' \Rightarrow$
 $d_\eta(n, n') + d_\eta(n', n'') = 1 + d_\eta(n', n'') > d_\eta(n, n'') = 1$
 since, by lemma 4.14.2, $d_\eta(n', n'') \in [0, 1]$

□

Definition 4.14.2: Let us suppose two SODTs $T = (V, E)$, $T' = (V', E')$ of the same order O where each node of T and T' has a 3-tuple $n_i = (q_i, \sigma_i, m_i) \in \Theta \times \mathbb{R} \times (0, +\infty)$ and $n'_i = (q'_i, \sigma'_i, m'_i) \in \Theta \times \mathbb{R} \times (0, +\infty)$ associated, respectively. We define the **SODT-distance** between T and T' as a function

$$\hat{\delta} : \hat{\Omega} \times \hat{\Omega} \rightarrow \mathbb{R}^+$$

$$\hat{\delta}(T, T') := \sum_{i=1}^O d_\eta(n_i, n'_i) := \sum_{i=1}^O (d_\eta((q_i, \sigma_i, m_i), (q'_i, \sigma'_i, m'_i)))$$

where d_η is the distance between nodes of a decision tree described in definition 4.14.1.

Proposition 4.14.2: Let us suppose that the same conditions as in the previous definition are fulfilled, then $\hat{\delta}$ is a metric on $\hat{\Omega}$. Thus, $(\hat{\Omega}, \hat{\delta})$ is a metric space.

Proof. Let us see that $\hat{\delta}$ satisfies the requirements of a metric:

a) $\hat{\delta}(T, T') \geq 0$ ($= 0 \Leftrightarrow n = n'$) (Not negativity) (identity of indiscernibles)

$$\hat{\delta}(T, T') = \sum_{i=1}^O d_\eta(n_i, n'_i) \geq 0$$

since $range(d_\eta) = [0, 1]$.

b) $\hat{\delta}(T, T'') \leq \hat{\delta}(T, T') + \hat{\delta}(T', T'')$ (Triangular inequality)

$$\begin{aligned} \hat{\delta}(T, T'') &= \sum_{i=1}^O d_\eta(n_i, n''_i) \\ &\stackrel{d_\eta \text{ is a metric}}{\leq} \sum_{i=1}^O (d_\eta(n_i, n'_i) + d_\eta(n'_i, n''_i)) \\ &= \sum_{i=1}^O d_\eta(n_i, n'_i) + \sum_{i=1}^O d_\eta(n'_i, n''_i) \end{aligned}$$

since d_η satisfies the triangular inequality.

$$\text{c) } \hat{\delta}(T, T') = \hat{\delta}(T', T) \quad (\text{Symmetry})$$

$$\hat{\delta}(T, T') = \sum_{i=1}^O d_\eta(n_i, n'_i) = \sum_{i=1}^O d_\eta(n'_i, n_i) = \hat{\delta}(T', T)$$

□

Note: It can be observed that, for $q_i, q'_i \in \Theta$, $\sigma_i, \sigma'_i \in \mathbb{R}$ and $m_i \in (0, +\infty)$, it is fulfilled that

$$d_\eta((q_i, \sigma_i, m_i), (q'_i, \sigma'_i, m_i)) = d_n((q_i, \sigma_i), (q'_i, \sigma'_i)),$$

where d_n is the distance that we introduced in section 3.17 between two nodes of a crisp decision tree and d_η is the distance between two nodes of a SODT that we have introduced in this section. That means that, in case the “unpreciseness” factor m_i is equal in two nodes of a SODT, then the η -distance between these two nodes will be equal to n -distance between the corresponding crisp nodes. Moreover, for any $m'_i \in (0, +\infty)$

$$d_\eta((q_i, \sigma_i, m_i), (q'_i, \sigma'_i, m'_i)) \geq d_n((q_i, \sigma_i), (q'_i, \sigma'_i)).$$

These observations lead us to the conclusion that the distance between two DTs that we proposed in section 3.17 and the distance between two SODTs proposed in this section are compatible.

Chapter 5

Crisp DT vs. SODT

In this chapter we are mainly concerned with the numerical comparison of the structural stability, the accuracy and the goodness functions of a crisp DT and our SODT model.

It is generally recognised that recursive partitioning, as used in the construction of classification trees, is inherently unstable, particularly for small data sets. Classification accuracy and, by implication, tree structure, are sensitive to changes in the training data. That is, tree classification algorithms may produce dramatically different rules if the training sample is slightly changed. The rules describing a decision tree are defined according to the splitting 2-tuples (or 3-tuples in the SODT model) associated to each node. Hence, the cause of the instability of tree classifiers is the instability of the split selection methods (see [38]). For this reason, we analyze them in the next section.

5.1 Split selection

As we have explained in chapters 3 and 4, the construction of tree structured classifiers consists in dividing the subsets of the measurement space by splitting them repeatedly into descendant subsets. In this section, we compare numerically the split selection methods of a crisp DT and a SODT. More concretely, we compare the change produced by a crisp decision tree and by a SODT in the tree structure (i.e., split change) when the learning data change. With this purpose, we create some noisy learning samples from a learning sample \mathcal{L} and analyze the variance of the sets of split points chosen by both models.

First of all, let us describe the learning samples we have used to run these simulations.

5.2 Learning data

Let (X, Y) be jointly distributed random variables with the q -dimensional vector X denoting a pattern or feature vector and Y denoting the associated class label of X . The components of X are the features.

Let us assume that X is of continuous ordered type; furthermore, let X take values from \mathbb{R}^q . Let Y take integer values $\{1, 2, \dots, J\}$, i.e., there are J classes of concern. Then

the goal of any classification scheme in general, and of SODT in particular, is to estimate Y based on the observation of X .

Example: simulated data

Let us suppose a learning data sample where the measurement space can not be partitioned and labeled (see section 3.13) in a way so that every object (see definition 3.3.1) is well classified, but where the measurement space has an overlapping region with objects of different classes mixed together. We will see how a crisp DT and an SODT partition this measurement space and how they behave when we introduce a certain noise to this learning data sample. Our purpose is to see if a SODT is structurally more stable than a crisp DT.

A two class classification problem with two features

Here we would like to analyze the particular case of a classification problem that distinguishes between two classes, i.e., considering the supervised learning from a statistical point of view, $Y \in \{0, 1\}$, with a two-dimensional feature vector $X = (X_1, X_2)$. Further, we suppose

$$X_1|Y = 0 \sim \mathcal{N}(\mu'_0, \sigma'),$$

$$X_1|Y = 1 \sim \mathcal{N}(\mu'_1, \sigma')$$

and

$$X_2 \sim \mathcal{N}(\mu'', \sigma''),$$

where $\mu'_0, \mu'_1, \mu'' \in \mathbb{R}$ and μ'_1 is calculated as a function of the 5% quantil of $\mathcal{N}(\mu'_0, \sigma')$ as follows:

Let $q_0 = \Phi^{-1}(0.05) + \mu'_0$, being Φ the distribution function of a $\mathcal{N}(0, 1)$. Let $q_1 = \mu_0 - \beta$, for a $\beta = 9/4 |\mu_0 - \mu_0|$. We choose μ'_1 in a way so that q_1 is the 95% quantil of $\mathcal{N}(\mu'_1, \sigma')$, i.e., so that $q_1 = (\Phi^{-1}(0.95)) \cdot \sigma' + \mu'_1$. (see figure 5.1).

The intention of this selection is to have a region in the middle where the objects of class 1 and objects of class 2 are mixed together and therefore a measurement space with difficulties to be partitioned into subsets with low impurity measures (see sections 3.6.4 and 4.7.3).

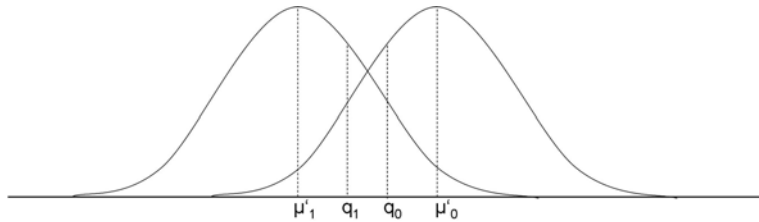


Figure 5.1: Quantils q_0 and q_1

Data simulation

Figure 5.2 is a simulation of one sample of $N = 50$ random values of the previously described random variable X . In this figure, the x-axis represents the variable X_1 and the y-axis the variable X_2 . The elements of the learning sample belonging to class 1 are represented with the symbol “○” and the ones belonging to class 0 with the symbol “*”.¹

We call this learning sample \mathcal{D} .

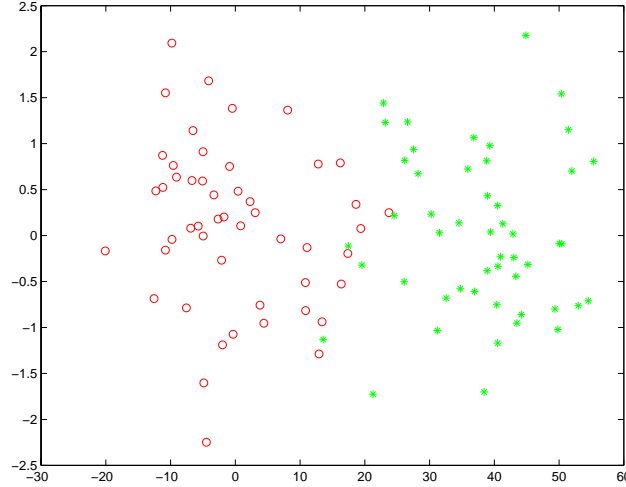


Figure 5.2: Learning sample \mathcal{D} . *: class 0, ○: class 1.

5.3 Split selection: goodness function

To build a tree classifier it is necessary to find at each internal node a split for the data. In sections 3.6.2 and 4.7.4 we explain the process of selection of the splits in the crisp DT and in the SODT case, respectively. Recall that the selected split is the one which maximizes a certain goodness function

$$\nu : (\{1, \dots, h\} \times \mathcal{X} \times (0, +\infty)) \times V(T) \longrightarrow \mathbb{R},$$

which is not the same in the crisp DT case as in the SODT case. The split point chosen in the crisp DT is a 2-tuple (q, σ) (see definition 3.4.5) and in the SODT case a 3-tuple (q, σ, m) (see definition 4.3.2).

Example

For our example \mathcal{D} , the variable chosen to split by both SODT and crisp DT is x_1 . In the right part of figure 5.3 we represent the goodness in the SODT (upper part) and crisp

¹In this sample, the quantils are $q_0 = 23.01$ and $q_1 = 16.27$.

(lower part) cases of the possible threshold values that divide the learning sample \mathcal{D} . The unpreciseness factor is $m = 1.07$. The threshold value is equal to $sp_1 = 19.54$ in the SODT case and equal to $sp_2 = 19.37$ in the crisp case, as it is shown in figure 5.5. In this figure, which we explain in detail in section 5.5, sp_1 and sp_2 are the y-coordinates corresponding to $x = 1$.

The left part of figure 5.3 represents the degree of membership (see section 4.4) μ_{n_L} and μ_{n_R} of the elements of \mathcal{D} to both left (blue points) and right (red points) child nodes, respectively. The upper part shows it in the SODT case and the lower part in the crisp DT.

Moreover, it can be seen that the SODT goodness function is differentiable, this fact is analyzed in section 5.12.

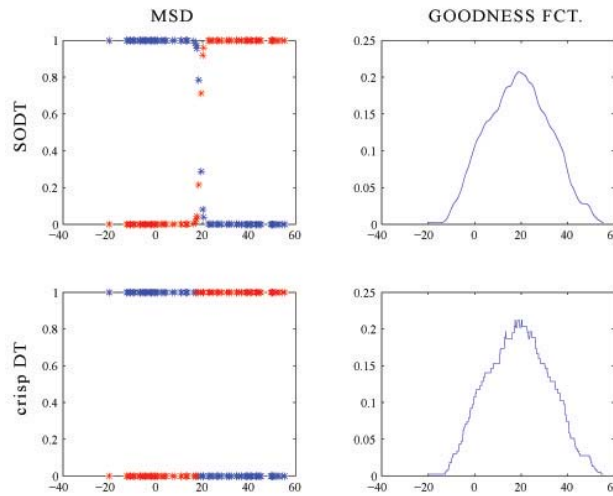


Figure 5.3: Goodness of the threshold values for the learning sample \mathcal{D}

5.4 Noise

Let $\mathcal{L} \subseteq \mathcal{X} \times \mathcal{C}$ be a learning sample and let x_i be the variable chosen to split. To test the stability of the SODT method in comparison with a crisp decision tree, we create some noisy data from the learning sample \mathcal{L} . This noise follows a normal distribution $\mathcal{N}(0, \sigma^2)$, where the variance σ^2 is the 5% of the mean value of the domain of definition of x_i .

Example

We add the previously described noise to the learning sample \mathcal{D} represented in figure 5.2. Since this noise is a random variable, every time we add it, the resulting noisy sample is different. An example of one of the possible noisy data \mathcal{D}' coming from \mathcal{D} can be seen in figure 5.4, which is the result of the addition of this noise once to the data in figure 5.2. The noise in this sample follows a normal distribution with variance $\sigma^2 = 0.94$.

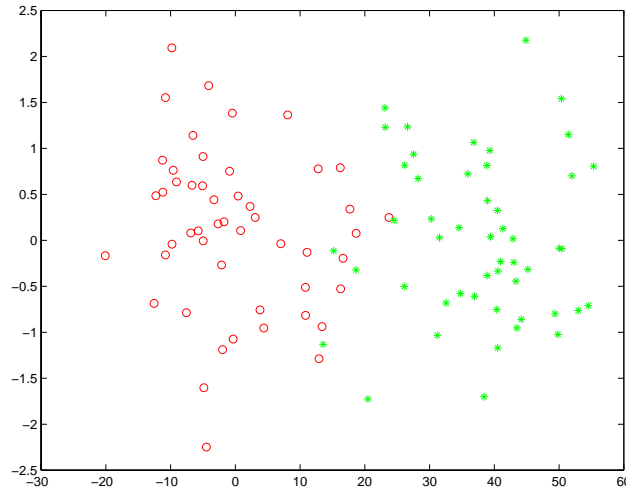


Figure 5.4: Learning sample \mathcal{D}' . *: class 0, \circ : class 1.

5.5 Preparation of sets for the study of variance

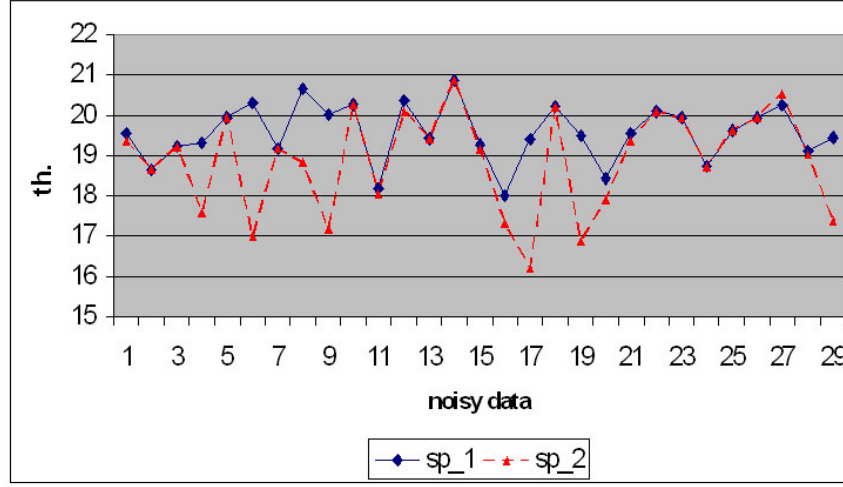
The process described in the previous section (5.4) will be repeated several times in order to observe the variance of the split points chosen for each data set. Our aim is to see if in the SODT case this variance is smaller than in the crisp case.

If we repeat this process of addition of noise to a learning sample \mathcal{L} N_D times, then we get N_D learning samples which only differ in the aleatory noise we introduce. For each of these databases, we calculate the chosen split point, by the SODT and by the crisp decision tree methods. Following this process, we will have the set S_1 of N_D split points obtained by SODT, and the set S_2 of N_D split points obtained by CRISP DT.

Example

In our example (where the original learning sample is \mathcal{D}), S_1 and S_2 would be the ones shown in figure 5.5. In this figure, the x-axis represents just an enumeration of the original and each of the noisy learning samples and the y-axis the threshold value selected to partition the measurement space of these learning samples. The threshold values corresponding to $x = 1$ are the ones for the original learning sample and $N_D = 28$.

In this example the variance s_1^2 of S_1 is equal to 0.50 and the variance s_2^2 of S_2 is 1.58. Hence $s_1^2 < s_2^2$, which means that the variance of the threshold values calculated with the SODT model is smaller than with the crisp one. In order to see that the difference between this two variances is statistically significant, we realize a study of the variance.

Figure 5.5: Split points chosen S_1 and S_2

5.6 Study of variance

We would like to test if the variance of the split points of the noisy data given by SODT method is smaller than the variance of the split points obtained by the crisp method. With this purpose, we will apply the F-test for the comparison of the standard deviations.

5.6.1 F-test of equality of two standard deviations

Suppose we have two normally distributed random variables with variances σ_1^2 , σ_2^2 . The hypothesis for the one-sided lower test are:

$$H_0 : \sigma_1 = \sigma_2$$

$$H_1 : \sigma_2 > \sigma_1$$

And the test statistic:

$$F = \frac{s_1^2}{s_2^2},$$

denoting s_1^2 , s_2^2 the sample variances. We will reject the null hypothesis for a significance level α if $F < F_{1-\alpha, N_1-1, N_2-1}$, being N_1 , N_2 the sample sizes. Thus, $F_{1-\alpha, N_1-1, N_2-1}$ is the critical value of the F-distribution with $N_1 - 1$, $N_2 - 1$ degrees of freedom and a significance level α , or, in other words, a confidence level $1 - \alpha$.

In the following section, we will illustrate the application of this test to the sets S_1 , S_2 .

5.6.2 Example. Variance of the split points: SODT vs. crisp DT

Here we will apply the F-test to the sets S_1 , S_2 (see section 5.5) of split points obtained from the SODT and crisp methods, respectively.

Thus, we first applied the Kolmogorov-Smirnov test and the Shapiro-Wilk tests for normality (see [62]). For both tests, we could not reject the hypothesis that S_1 and S_2 are normally distributed for a significance level $\alpha = 0.005$.

The values of the standard deviations for S_1 and S_2 are $s_1 = 0.71$ and $s_2 = 1.25$, respectively.

And thus, the test statistic is

$$F = \frac{s_1^2}{s_2^2} = 0.32$$

Taking into account that the number of elements of S_1 is equal to the number of elements of S_2 and equal to 29, then we have $F = 0.32 < 1.88 = F_{0.95,28,28}$.

After that, the hypothesis H_0 that the two standard deviations are equal is rejected (for a lower one tailed test).

Consequently, in this case we prefer SODT to crisp DT because the variance of the split points chosen is smaller (the variable chosen is the same in both cases and the threshold values variance is statistically significantly smaller for our model).

5.7 Example: Simulated data \mathcal{D}_2

In this section we simulate a learning sample \mathcal{D}_2 with the same procedure described in section 5.2 for $N = 500$. A representation of these data can be seen in figure 5.6. The quantils are in this case $q_0 = 13.51$ and $q_1 = 17.88$.

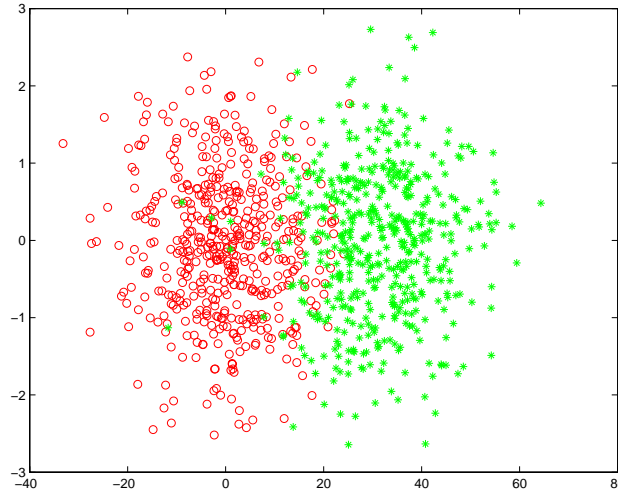
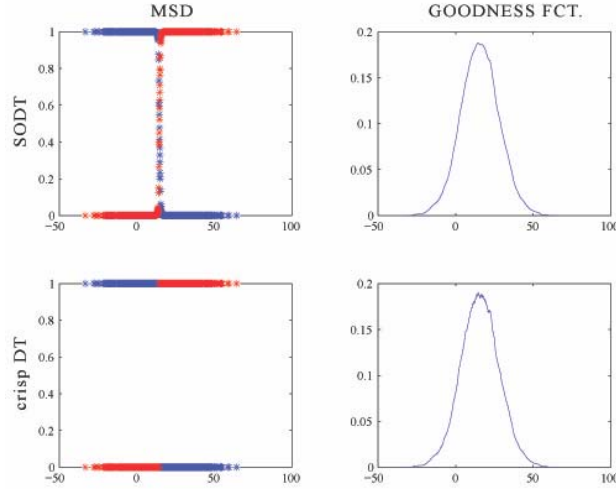


Figure 5.6: Learning sample \mathcal{D}_2 . *: class 0, ○: class 1.

The goodness function is represented in figure 5.7 in the same form as described for the previously example \mathcal{D} .

In this case, the unpreciseness factor associated to the node we are splitting is $m = 1.64$ and the variable chosen to split is in both, the crisp and the SODT case, x_1 . The threshold

Figure 5.7: Goodness of the threshold values for the learning sample \mathcal{D}_2

value is the value which maximizes the goodness function, which in the SODT case is equal to $sp_1 = 14.96$ and in the crisp DT case is equal to $sp_2 = 15.26$.

The noise (see section 5.4) follows a $\mathcal{N}(0, \sigma^2)$, where $\sigma^2 = 0.77$. And the procedure of adding noise to \mathcal{D}_2 is repeated $N_D = 100$ times. We obtain thus 100 noisy learning samples and calculate the threshold values for each of them, by a SODT and a crisp DT.

Let us denote S_1^2 and S_2^2 the sets of threshold values obtained by SODT and crisp DT, respectively. The variance of S_1^2 is $s_1^2 = 0.50$ and the variance of S_2^2 is $s_2^2 = 0.74$, i.e., $s_1^2 < s_2^2$, which means that the variance of the threshold values calculated with the SODT model is smaller than with the crisp one.

We test now if the difference between these two variances is statistically significant.

First, we apply the Kolmogorov-Smirnov test and the Shapiro-Wilk tests for normality (see [62]) and obtain that we could not reject the hypothesis that S_1^2 and S_2^2 are normally distributed for a significance level $\alpha = 0.005$.

F-test

The cardinality N_1 of S_1^2 and N_2 of S_2^2 is

$$N_1 = N_2 = 101$$

The test statistic is

$$F = \frac{s_1^2}{s_2^2} = \frac{0.5031}{0.7400} = 0.67$$

and the critical value of the F-distribution required for this test is $F_{0.95,100,100} = 1.39$. Thus, the condition to reject the hypothesis H_0 with a confidence level of 0.95 is fulfilled:

$$F = 0.67 < 1.39 = F_{0.95,100,100}.$$

To summarize, in case of randomly distributed data with an overlapping region, SODT is preferable because the variance of the split points is smaller than in the usual crisp DT.

5.8 Example: Credit rating data

In this section we will realize the same process described in previous section for a learning sample which is not simulated by us, but coming from credit rating. The data \mathcal{F} we are presenting are calculated in [54] as score of the credit rating data which can be found in [22]. It is a learning sample of size $N = 1000$ and its representation can be seen in figure 5.8.

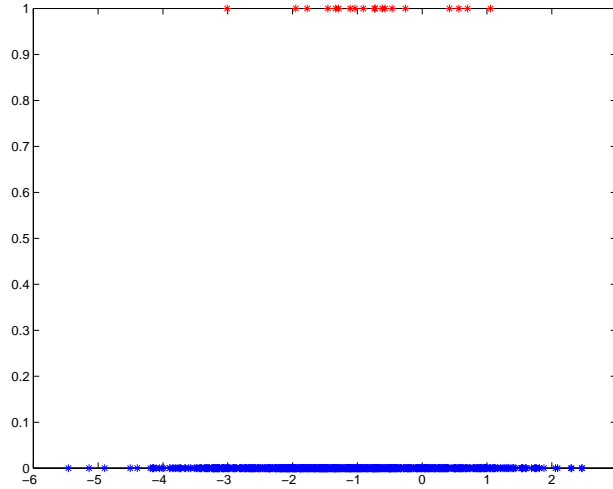


Figure 5.8: Learning sample \mathcal{F}

Since the measurement space of this sample is one-dimensional, the variable chosen to split is the unique variable x_1 . The unpreciseness factor chosen by our algorithm is $m = 3.60$.

The representation of the goodness function of every possible threshold value in the crisp and SODT cases can be seen in figure 5.9.

From this learning sample \mathcal{F} , we create 30 noisy learning samples as we explained in section 5.4. For each of these noisy samples, we calculate the chosen split point, by the SODT and by the crisp decision tree methods. Let us call $S_1^{\mathcal{F}}$ the set of threshold values obtained by SODT and $S_2^{\mathcal{F}}$ the set of threshold values obtained by crisp DT. As an illustration of them, see figure 5.10. In this figure, the x -axis represents the enumeration of each of the noisy learning samples and the y -axis the threshold value selected to partition the measurement space of these learning samples.

The variance of $S_1^{\mathcal{F}}$ is $s_1^2 = 0.017$ and the variance of $S_2^{\mathcal{F}}$ is $s_2^2 = 0.063$. Hence $s_1^2 < s_2^2$, i.e., the variance of the threshold values calculated with the SODT model is smaller than with the crisp one.

Let us see if this difference is statistically significant.

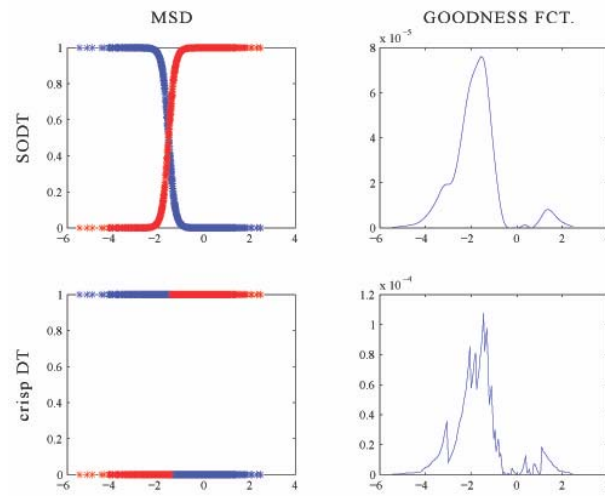


Figure 5.9: Goodness of the threshold values for the learning sample \mathcal{F} .

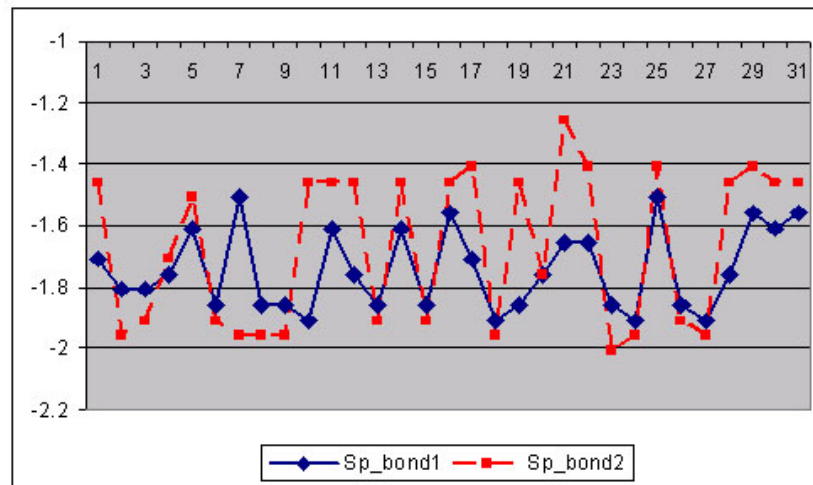


Figure 5.10: Split points for data \mathcal{F} with noise. SODT: blue, crisp DT: red

F-test

The number of elements of $S_1^{\mathcal{F}}$ and of $S_2^{\mathcal{F}}$ is

$$N_1 = N_2 = 31$$

The test statistic is

$$F = \frac{s_1^2}{s_2^2} = \frac{0.0175}{0.0635} = 0.27$$

where s_1 and s_2 are the standard deviations of $S_1^{\mathcal{F}}$ and $S_2^{\mathcal{F}}$, respectively. The critical value of the F-distribution with $N_1 - 1$, $N_2 - 1$ degrees of freedom and a confidence level 95% is

$$F_{0.95,30,30} = 1.84$$

and thus

$$F = 0.27 < 1.84 = F(0.95, 30, 30)$$

which means that the hypothesis H_0 that the two standard deviations are equal is rejected with a confidence level of 0.95.

5.9 Conclusion

To summarize, we have investigated how two models react to a change (following a $\mathcal{N}(0, \sigma^2)$, see section 5.4) in the learning sample. We see numerically that the resulting threshold value change is statistically significantly smaller for the SODT than for the crisp DT in cases where the measurement space is one-dimensional or when the variable chosen to split is the same for all noisy learning samples.

Consequently, in these cases SODT is preferable to crisp DT because the variance of the split points chosen is smaller (the variable chosen is the same in both cases and the threshold values variance is statistically significantly smaller for our model), which indicates that our model is less sensitive to noise.

5.10 Distance between DTs and between SODTs

In case the addition of noise to the learning sample does not only imply a change on the threshold value chosen to split, but also on the split variable, this difference needs to be considered. For this purpose, we analyze the distance between decision trees (DT) and between SODTs that we introduced in sections 3.17 and 4.14. Our aim is to compare numerically the sensibility to noise of a DT and of a SODT in this type of learning sample .

For the numerical test we consider a simulated learning sample and a learning sample coming from medicine. We add some noise to the learning sample \mathcal{L} and obtain a learning sample \mathcal{L}' . Let T_{DT} and T'_{DT} be the DT learned from \mathcal{L} and \mathcal{L}' , respectively. Furthermore,

let T_{SODT} and T'_{SODT} be the SODT learned from \mathcal{L} and \mathcal{L}' , respectively. Our goal is to observe if the distance between T_{DT} and T'_{DT} is bigger than the distance between T_{SODT} and T'_{SODT} , which would mean that SODT is less sensitive to noise than DT.

5.10.1 Simulated Data

From the simulated learning sample \mathcal{L} of figure 5.11, we have created 30 noisy learning samples as it is explained in section 5.4. Figure 5.12 represents an example of one of the 30 noisy learning samples. For each of them we have calculated the resulting DT and SODT and obtained the distance as explained at the beginning of this section.

In figure 5.13, the x-axis indicates an enumeration of the noisy learning samples, and the y-axis indicates the distance between the trees learned from the original and from the noisy learning sample. We observe that the distances for the SODT case are smaller than the distances for the crisp case. The sample mean of all these distances in the crisp case is $\bar{y}_{DT} = 0.4506$ while in the SODT case it is $\bar{y}_{SODT} = 0.3025$.

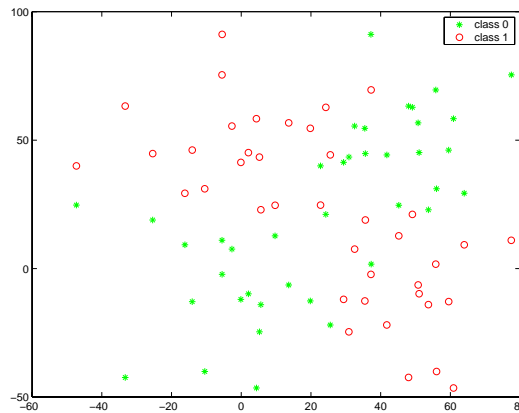


Figure 5.11: Learning sample \mathcal{L} .

Heart disease dataset

The learning sample for this numerical test is coming from the medicine, in concrete from the analysis of heart disease ([15]). This learning sample that we call “ He ” consists of 270 observations. Each of these observations is described by 13 attributes and 2 possible outcome classes, class 0 represents the presence of heart disease and class 1 the absence of it. From these observations, the 55.56% of them belong to class 0 and the 44.44% to class 1.

From this learning sample He we have created 10 noisy learning samples as it is explained in section 5.4. Next, for each of them we have calculated the resulting DT and SODT and obtained the distance as explained before.

In figure 5.14, the x-axis indicates an enumeration of the noisy learning samples, and the y-axis indicates the distance between the trees learned from the original and from the

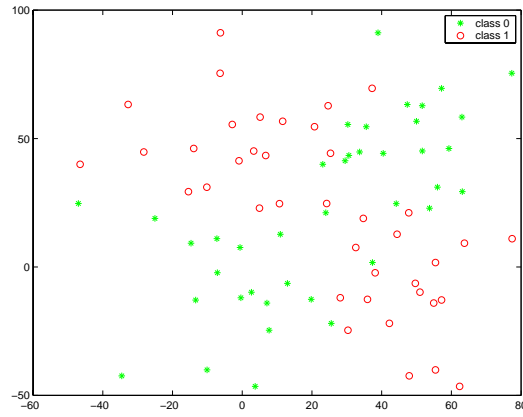
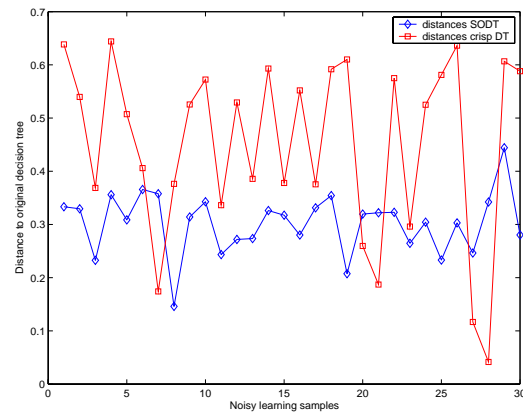
Figure 5.12: Learning sample \mathcal{L}' .

Figure 5.13: Distance between original tree classifier and tree classifier coming from noisy learning samples

noisy learning sample. We observe that the distances for the SODT case are smaller than the distances for the crisp case. The mean in the crisp case is equal to $\bar{y}_{DT} = 0.5571$ and in the SODT case it is $\bar{y}_{SODT} = 0.4236$.

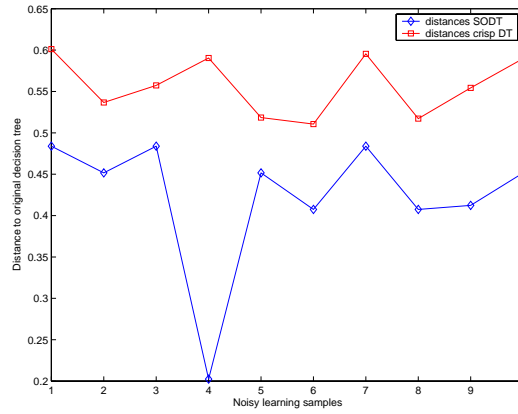


Figure 5.14: Distance between trees learned from noisy data and the original tree coming from He

5.10.2 Conclusion

To summarize, we have investigated, by analyzing distances between trees, how two models react to a change (following a $\mathcal{N}(0, \sigma^2)$, see section 5.4) in the multidimensional learning sample. We see numerically that the distance between the crisp DT learned from the original learning sample and the crisp DT learned by a noisy learning sample is bigger than the distance between the SODT learned from the original learning sample and the SODT learned by a noisy learning sample.

Since this distance is proportional to the instability coefficient (see section 3.18), we conclude that for the numerical simulations our model is less sensitive to noise than the crisp DT and therefore more stable.

5.11 Accuracy

In this section, we compare numerically the degree of conformity of the learning sample to the classification obtained by the crisp DT and the SODT model. To compare this accuracy, the misclassification rate is calculated.

Let (X, Y) be jointly distributed random variables with h -dimensional vector X denoting a feature or measurement vector that takes values from \mathcal{X} and Y denoting the associated class label of X . Y takes values from $\mathcal{C} = \{1, \dots, J\}$.

Let $d : \mathcal{X} \rightarrow \mathcal{C}$ be a classifier, i.e., $d(\cdot)$ will estimate Y based on observing X . Recall that the **true misclassification rate** of d , denoted by $R^*(d)$, is

$$R^*(d) = P(d(X) \neq Y).$$

where P denotes the probability (see the description in definition 3.14.2 of a probability model).

Conceptually, as we pointed out in chapter 3, the process of obtaining the true misclassification rate $R^*(d)$ of a classifier which is created from a learning data set would be:

- Construct $d(\cdot)$ using \mathcal{L} .
- Draw a virtually infinite set of cases from the same population as \mathcal{L} was drawn from.
- Observe the correct classification for each of these cases.
- Find the predicted classification using $d(\cdot)$.
- $R^*(d)$ is the proportion of misclassifications by d .

There is no difficulty to calculate $R^*(d)$ if the learning sample \mathcal{L} consists of simulated data following a certain distribution function. Since in most of the cases only the data in \mathcal{L} are available with no possibility of getting an additional large example of classified measurement vectors, the true misclassification rate is estimated and \mathcal{L} must be applied both to construct the function $d(\cdot)$ and to estimate $R^*(d)$. The estimation of the true misclassification rate can be done in several ways. In section 3.14, we describe the resubstitution estimate, the test sample estimate and the V-fold cross validation estimate. For the implementation of our method, we choose the test sample estimate and call it “ MR ”. The reason why we do not use resubstitution estimate is because it can lead to overfitting due to the fact that the data sample used to learn is the same as the one used to test. Neither V-fold cross validation estimate is chosen, since we want to compare the tendency to overfitting of the SODT and crisp DT models, basically due to the differences of the goodness functions of these two models, and this estimate itself is normally used as a method to reduce overfitting (see section 3.14).

As stopping rule, we chose the “two stage search” (see section 3.12). At the first stage, we determine the structure of the tree. Since we are considering binary rooted trees, the structure is uniquely determined by fixing its level (see definition 2.7.7). At the second stage, we look for the splits of all nodes with the crisp DT and SODT models and compare the resulting estimated misclassification rates. We select this stopping rule in order to compare the evolution and the overfitting tendency of the two models (the larger the tree, the more it fits the learning sample).

Following, we test numerically the accuracy and overfitting tendency of our model in comparison with a crisp decision tree. For the numerical tests we take the simulated learning sample “ \mathcal{L} ” and the learning sample “ He ”, which were introduced in sections 5.10.1 and 5.10.1, respectively.

5.11.1 Simulated data

For the first of these numerical tests we take the simulated learning sample “ \mathcal{L} ” of section 5.10.1.

Since we are going to compute the test sample estimate, we divide the data sample into two: training sample and test sample, where the training sample is taken as the $2/3$ of the complete data sample and the test sample the rest. In figure 5.15, the x-axis represents the first stage of the tree induction, i.e., as we have explained previously, the fixed level of the tree. The y-axis represents the MR . In this figure, we can see the MR values for both the learning and test set computed by the SODT and by the crisp DT models.

We can observe that, for the \mathcal{L} data sample, the MR at the root node ($x = 1$) of a crisp DT and a SODT is obviously the same, for both learning and test sets. For each fixed level $x = l \in \{2, \dots, 6\}$ of the tree, then, the SODT model MR of the learning sample is bigger or equal than in the crisp case, but the MR of the test sample is smaller than in the crisp case. In general we conclude that, for this example, the SODT model is more accurate and has less tendency to overfit the learning sample than the crisp mode

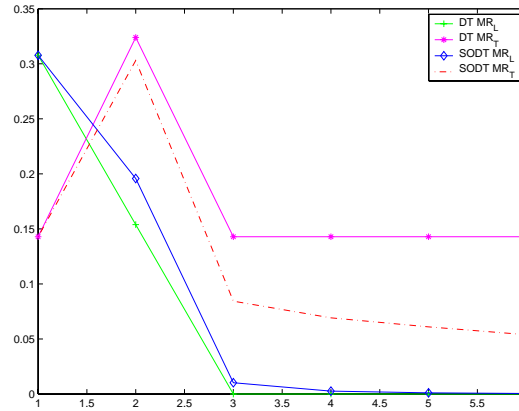


Figure 5.15: MR \mathcal{L} . DT MR_L: MR learning set (DT), DT MR_T: test set (DT), SODT MR_L: MR learning set (SODT), SODT MR_T: MR test set (SODT)

5.11.2 Medical diagnosis data

Medical diagnosis is one of the fields where the trade off between accuracy and interpretability plays an important role. Since thousands of data are collected from patients, these data, in combination with expert knowledge, are used to create a model that fits them. This model should be able to make a diagnosis to a new patient and thereby, the model should not adjust too much to the data used for learning, i.e., should not overfit the data. Another important aspect is the interpretability (see [40]) of this model, because medical experts do not only want to learn about the problem of interest, but they also want to check whether the knowledge represented within the learning system is correct or

at least plausible. For this reason, decision trees, which are white box models, are a good choice for supervised learning from medical datasets.

Example: Heart disease dataset

In this section, we test numerically the accuracy and overfitting tendency of our model in comparison with a crisp decision tree for the learning sample “*He*”, which is coming from medicine and was introduced in section 5.10.1.

In order to compute the test sample estimate, we divide the data sample into two: training sample and test sample, where the training sample is taken as the 2/3 of the complete data sample and the test sample the rest.

In figure 5.16, the x-axis represents the first stage of the tree induction, i.e., as we have explained previously, the fixed level of the tree. The y-axis represents the MR . In this figure, we can see the MR values for both the learning and test set computed by the SODT and by the crisp DT models.

We can observe that, for the *He* data sample, the MR at the root node ($x = 1$) of a crisp DT and a SODT is obviously the same, for both learning and test sets. For each fixed level $x = l \in \{2, \dots, 6\}$ of the tree, then, the SODT model MR of the learning sample is bigger than in the crisp case, but the MR of the test sample is smaller than in the crisp case. In general we conclude that, also for this example, the SODT model is more accurate and has less tendency to overfit the learning sample than the crisp model.

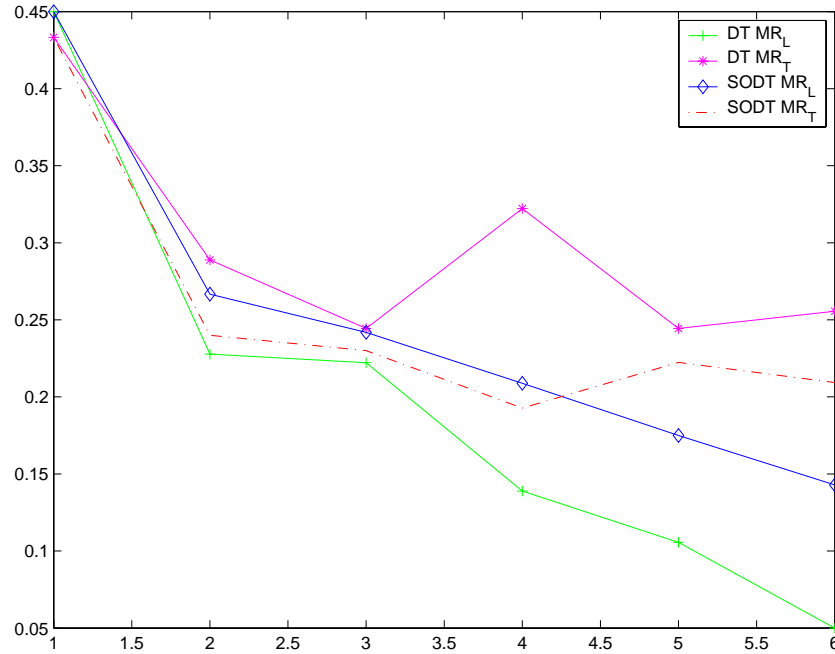


Figure 5.16: MR *He*. DT MR_L: MR learning set (DT), DT MR_T: test set (DT), SODT MR_L: MR learning set (SODT), SODT MR_T: MR test set (SODT)

Let us consider a simpler dataset and observe if the accuracy and tendency to overfitting also improves with the SODT model.

5.11.3 Example: \mathcal{D}_3

Let us consider the simple learning sample \mathcal{D}_3 presented in figure 5.17 together with the FL operator. As additional information, the goodness function, as well as the MSD to left and right child nodes of the root node, can be seen in figure 5.18.

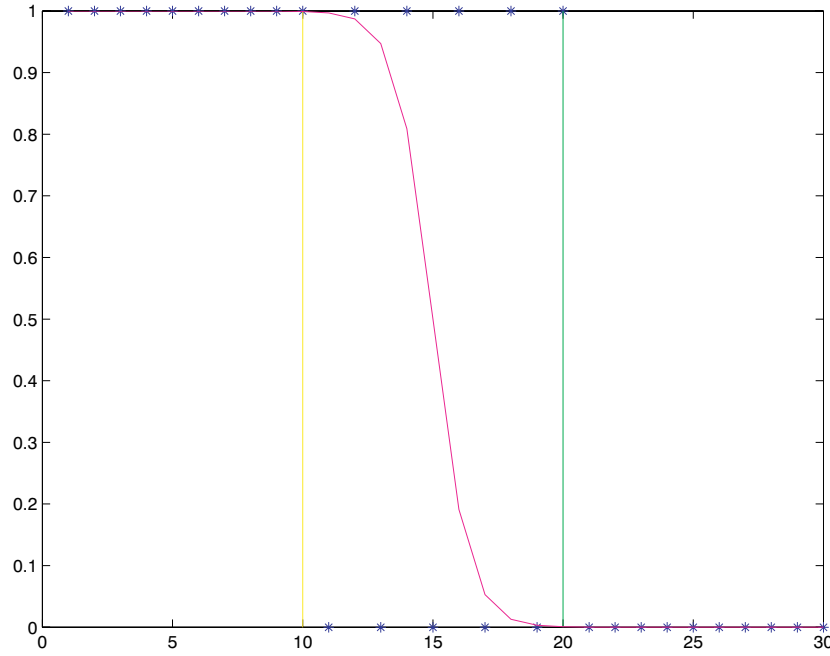


Figure 5.17: FL operator at the root node of a SODT constructed to learn from \mathcal{D}_3

In figure 5.19, the x-axis represents the first stage of the tree induction, i.e., the fixed level of the tree and the y-axis represents the MR . In this figure, we can see the MR values for both the learning and test set computed by the SODT and by the crisp DT models.

For the \mathcal{D}_3 data sample, the MR at the root node ($x = 1$) of a crisp DT and a SODT is the same, for both learning and test sets. For each fixed level $x = l \in \{2, \dots, 6\} \setminus \{4\}$ of the tree, then, the SODT model MR of the learning sample is bigger than in the crisp case, but the MR of the test sample is, except for level $x = 4$, smaller than or equal in the crisp case. We conclude that, for \mathcal{D}_3 , the SODT model is not much more accurate but has less tendency to overfit the learning sample than the crisp model since the MR of the training and test data sample in the SODT differ, at every fixed level $x = l \in \{1, \dots, 6\}$, less than in the crisp DT.

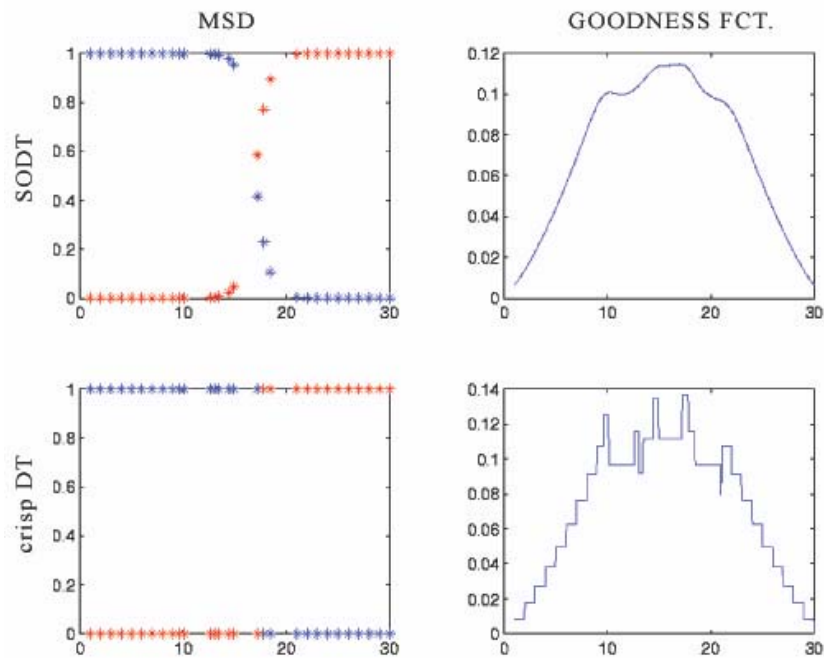


Figure 5.18: Goodness of the threshold values for the learning sample \mathcal{D}_3 (right part)(top part: SODT, bottom part: crisp DT) and membership degree to the left and right child nodes of the root node (left part)

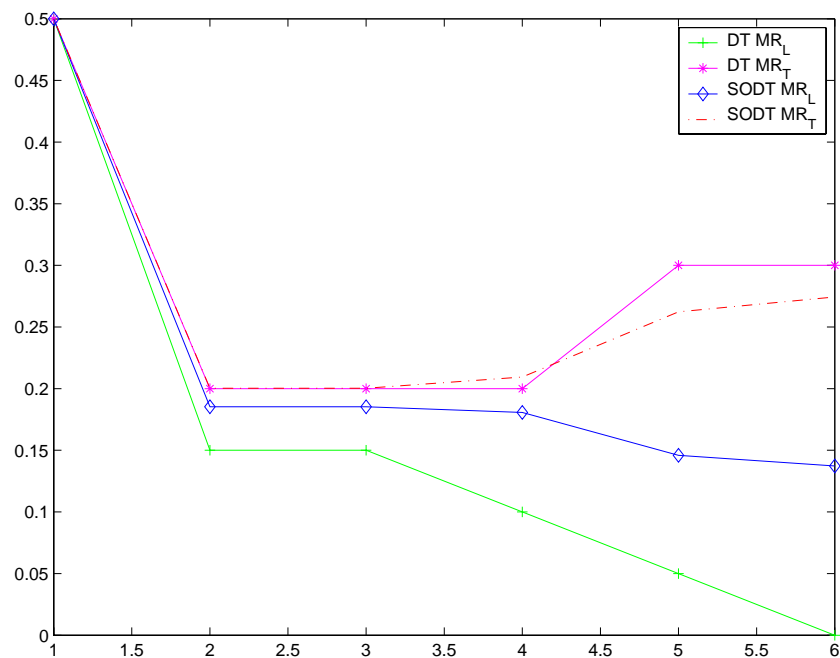


Figure 5.19: \mathcal{D}_3 . MR: Misclassification Rate. DT MR_L: MR learning set (DT), DT MR_T: test set (DT), SODT MR_L: MR learning set (SODT), SODT MR_T: MR test set (SODT)

5.12 Differentiability of function ν

Proposition 5.12.1: The SODT goodness function ν , which is introduced in definition 4.7.6,

$$\nu : (\{1, \dots, h\} \times \mathcal{X} \times (0, +\infty)) \times V(T) \longrightarrow \mathbb{R}$$

$$\nu((q, \sigma, m), n) = \Delta\tau(n, m_P) = \tau(n, m_P) - \rho_{m, n_L}\tau(n_L, m) - \rho_{m, n_R}\tau(n_R, m)$$

is differentiable w.r.t. the threshold value σ and w.r.t. the unprecisenes factor m .

Proof. This function is differentiable since it is a composition of the differentiable functions τ , ρ , μ and \tanh in their domains of definition. Then, $\frac{\partial \nu((q, \sigma, m), n)}{\partial m}$ is trivially computed by applying the chain rule as follows:

$$\begin{aligned} \frac{\partial \nu}{\partial m} &= \frac{\partial \tau(n, m_P)}{\partial m} - \frac{\partial \rho_{m, n_L}}{\partial m} \cdot \tau(n_L, m) - \rho_{m, n_L} \frac{\partial \tau(n_L, m)}{\partial m} \\ &\quad - \frac{\partial \rho_{m, n_R}}{\partial m} \cdot \tau(n_R, m) - \rho_{m, n_R} \frac{\partial \tau(n_R, m)}{\partial m}, \end{aligned}$$

where

$$\frac{\partial \tau(n, m_P)}{\partial m} = 0$$

and n_L and n_R represent the left and right child nodes of n respectively. Let us calculate the partial derivatives $\frac{\partial \rho_{m, n_L}}{\partial m}$ and $\frac{\partial \tau(n_L, m)}{\partial m}$ of the previous equation for the left node:

$$\frac{\partial \rho_{m, n_L}}{\partial m} = \frac{1}{P_n(X)} \cdot \frac{\partial}{\partial m} \left(\sum_{o \in X} \mu_{n_L}(o, m) \right) = \frac{1}{P_n(X)} \sum_{o \in X} \left(\frac{\partial}{\partial m} \mu_{n_L}(o, m) \right),$$

where

$$\frac{\partial \mu_{n_L}(o, m)}{\partial m} = \frac{\partial FL(o_q, \sigma, m)}{\partial m} \cdot \mu_n(o, m_P) \quad (5.1)$$

and

$$\frac{\partial FL(o_q, \sigma, m)}{\partial m} = \frac{1}{2} (\text{sech}(\sigma - o_q) \cdot m)^2 (\sigma - o_q). \quad (5.2)$$

$$\begin{aligned} \frac{\partial \tau(n_L, m)}{\partial m} &= \frac{\partial}{\partial m} \left(\sum_{i \neq j} \rho_m(i|n_L) \rho_m(j|n_L) \right) \\ &= \sum_{i \neq j} \frac{\partial \rho_m(i|n_L)}{\partial m} \cdot \rho_m(j|n_L) + \rho_m(i|n_L) \frac{\partial \rho_m(j|n_L)}{\partial m}, \end{aligned}$$

where

$$\begin{aligned}
\frac{\partial \rho_m(i|n_L)}{\partial m} &= \frac{\partial}{\partial m} \left(\frac{\sum_{(o,j) \in \mathcal{L}} \mu_{n_L}(o, m)}{\sum_{o \in X} \mu_{n_L}(o, m)} \right) \\
&= \frac{\sum_{(o,j) \in \mathcal{L}} \frac{\partial \mu_{n_L}(o, m)}{\partial m} \cdot \sum_{o \in X} \mu_{n_L}(o, m) - \sum_{o \in X} \frac{\partial \mu_{n_L}(o, m)}{\partial m} \cdot \sum_{(o,j) \in \mathcal{L}} \mu_{n_L}(o, m)}{\left(\sum_{o \in X} \mu_{n_L}(o, m) \right)^2}
\end{aligned}$$

For the right node the partial derivatives are calculated in the same way, just by changing equation 5.1 to

$$\frac{\partial \mu_{n_R}(o, m)}{\partial m} = \frac{\partial FG(o_q, \sigma, m)}{\partial m} \cdot \mu_n(o, m_P),$$

where

$$\frac{\partial FG(o_q, \sigma, m)}{\partial m} = \frac{1}{2} (\text{sech}(o_q - \sigma) \cdot m)^2 (o_q - \sigma) \quad (5.3)$$

Analogously, the $\frac{\partial \nu((q, \sigma, m), n)}{\partial \sigma}$ is calculated and the main difference is in equations 5.2 and 5.3, which in this case are

$$\frac{\partial FL(o_q, \sigma, m)}{\partial \sigma} = \frac{1}{2} (\text{sech}(\sigma - o_q) \cdot m)^2 \cdot m$$

and

$$\frac{\partial FG(o_q, \sigma, m)}{\partial \sigma} = -\frac{1}{2} (\text{sech}(o_q - \sigma) \cdot m)^2 \cdot m$$

respectively.

Note: In figure 5.20 it can be seen that the usual crisp goodness function is not differentiable w.r.t. the threshold value. The differentiability of our SODT goodness function allows the use of gradient descent optimization methods to calculate the optimal threshold value at each node of the tree.

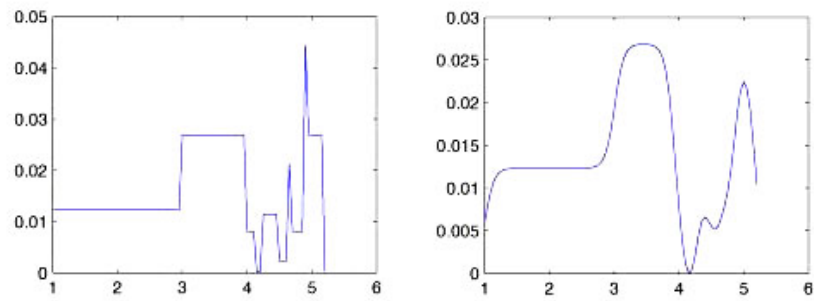


Figure 5.20: Example of goodness functions of the split thresholds. Left: crisp DT, right: SODT.

Chapter 6

Summary and conclusions

In the current work, binary tree classifiers are analyzed and partially improved. The analysis of tree classifiers goes from their topology from the graph theory point of view to the creation of a new tree classification model by means of combining decision trees and soft comparison operators with the purpose to not only overcome the well known instability problem (see [38]) of decision trees, but also in order to confer the ability of dealing with uncertainty.

The main results of chapter 2 are the theorems about division of a tree into levels and the enumeration of its nodes, which is used in the following chapters such as in chapter 3 in order to propose a formal definition of a decision tree. Furthermore, in chapter 3, we establish an order on the categorical variables so that splitting methods based on the election of a variable and a threshold to partition the measurement space can be applied. The process of learning a decision tree is presented, as well as its three main parts: induction, stopping rule and terminal node class assignation.

Further on, in section 3.16 we contribute to the DT inference process, which consists of the prediction of a class label to unclassified new instances, by offering an explicit formulation of the classifier/classification function represented by a decision tree. At the end of this chapter we propose an instability coefficient which is based on the notion of Lipschitz continuity.

Our most relevant contributions at this point are offering a metric to measure the proximity between decision trees together with the formal definition of decision tree and the explicit formulation of the inference process.

This thesis converges towards its main part in chapter 4, where we propose SODT, a new model that combines soft comparison operators and decision trees in order to avoid the structural instability of the latter, as well as to cope with uncertainty. The basis for this combination is a function μ defining the degree of membership of an element to a node. Based on this function, we define a probability measure of a subset A of the measurement space \mathcal{X} at a node $n \in V(T)$, which we call “fuzzy cardinality” of A at n . In section 4.4.2, by means of the fuzzy cardinalities, this thesis leads to theorems that prove that our SODT creates a fuzzy partition of each node and of the complete measurement space. Our concern with the asymptotical behaviour of function μ leads us to the conclusion that, when the

unpreciseness factor tends to infinity, our SODT tends to a crisp DT. Once this function, that will be the basis of our work, is analyzed, we depict the learning and inference process of a SODT.

The induction, stopping rule, terminal node class assignment and inference processes are modified in order to be adapted to this new model. Among them, the main modification is done in the induction process, where we propose, based on the fuzzy cardinalities, a probability function of a class $j \in \{1, \dots, J\}$ to each node of the tree as well as further functions which are needed for the creation of a tree classifier. Usual tree classifiers partition the measurement space by choosing those splits which maximize a certain goodness function (based on the Gini index, the information Gain function, ...) and the domain of definition of the goodness functions are the elements of the simplex Δ^{J-1} . Since our probability functions are elements of the simplex, similar formulas can be applied.

In case that variable combinations are more suitable than univariate splits, the new model is easily adapted since the fuzzification is done in the operators and thus, just by slightly changing the usual *FL* and *FG* soft comparison operators, multivariate splits are considered (see section 4.7.6).

Also of interest is the fact that this model can deal with data given with uncertainty, i.e., fuzzified data (see [36]) and the manner in which it is done is explained in section 4.11.

In figure 6.1 the influence the soft comparison operators *FG* and *FL* on the partition of the measurement space is outlined. By the inclusion of these operators, the crisp partition turns into a fuzzy partition, which indicates that the predicted classification can be given with uncertainty instead of determining a single class for any given instance. This use of overlapping class definitions improves the classification, as well as the interpretability of the results by providing more insight into the classifier structure.

Besides, this type of fuzzification through the comparison operators, provides the possibility to deal directly with real valued data (numerical variables) instead of transforming them into multidimensional linguistic variables for posterior fuzzification.

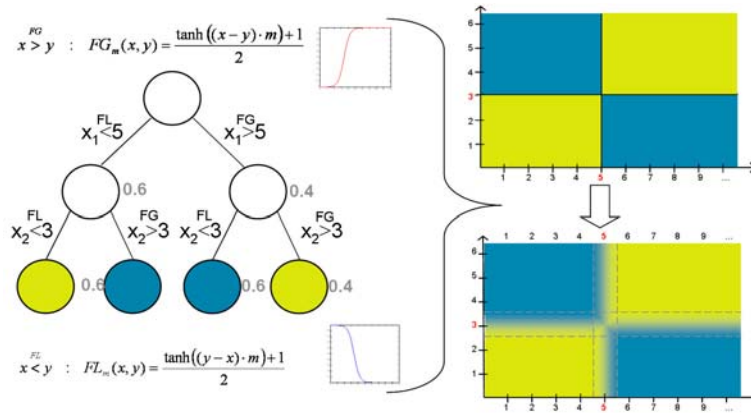


Figure 6.1: From crisp to fuzzy partition of the measurement space by means of soft comparison operators

In chapter 5 the results of the implementation of SODT are presented and compared

with the results obtained by a crisp DT in terms of the variance of the splits thresholds for noisy data, of distance between the tree classifiers and of accuracy of both models.

To summarize, SODT is preferred since we have investigated that this fuzzification results in a more stable tree classifier. If a number of changes is applied to a dataset, the variance of the split thresholds is statistically significantly smaller. We have analyzed this difference for several simulated data with an overlapping region and for data coming from a credit rating. Besides, the sample means of the distances between the resulting tree classifiers is also smaller in the SODT case than in the crisp case. The analysis of these distances has been done for simulated data and for data coming from medical diagnosis.

The accuracy is also analyzed and a smaller tendency to overfitting the training data with SODT than with crisp DT is observed.

The main drawback of this model is the introduction of a new parameter, the unpreciseness factor m , which is responsible of the fuzzification of the operators and it increases the complexity of the model. This model performs specially well when the data are given with uncertainty, noise or with overlapping regions. However it is not worthwhile to use this new model if none of these is the case.

On the other hand, the inclusion of soft operators allows the use of gradient descent methods for looking for the splitting threshold. Thus, in case large datasets are provided, this model can make the task of finding the threshold which maximizes the goodness function

$$\nu : (\{1, \dots, h\} \times \mathcal{X} \times (0, +\infty)) \times V(T) \longrightarrow \mathbb{R}$$

easier.

Further work

In the frame of this work, a model for the induction of a tree classifier with a differentiable goodness function was created. This fact can be used to apply gradient descent methods for the optimization of its parameters. These methods would find the parameters which maximize the goodness function without the need of computation of this function at each possible split point.

Further work could lie in the application of a gradient descent algorithm for optimizing the goodness function.

Another further line of investigation would be the application of the basics of our SODT for other purposes instead of classification. Actually, some interesting work is being done in this direction. Parallel to this thesis, Jan Hauth [30], a colleague of the author, has used the main idea of SODT for regression tasks. His aim is the identification of non-linear dynamical systems. For this purpose, he considers a partition of the measurement space and identifies a linear system on each set of the partition. This partition is represented in form of a tree. In order to get a global non-linear system, the partial linear systems are combined by means of smooth weight functions. These weight functions actually correspond to our function μ_n , which denotes the degree of membership to each node n of the tree.

Appendix A

Fuzzy sets

Several authors in the literature ([13], [6], [40] and others) explain that in many situations the assumption of crisp membership or nonmembership of an object or element x to set A is too restrictive.

Contrary to a classical set a fuzzy set is a model of such a collection of objects in which an object needs not necessarily belong or not belong to this collection. It means that the transition from "belong to a set" ($x \in A$) to "not belong to a set" ($x \notin A$) is gradual rather than crisp. Such a transition is usually characterized by membership functions ranging from zero to one.

Fuzzy sets proposed by Zadeh (1965) are uniquely described by their membership functions. Using membership function we can specify a fuzzy set as follows:

$$\mu_A : X \rightarrow [0, 1], \text{ or } A : X \rightarrow [0, 1].$$

A fuzzy set A in X is directly specified by the function $\mu_A(x)$ (or $A(x)$) or indirectly by a set of ordered pairs $\mu_A(x)$ (or $A(x)$) where $\mu_A(x)$ (or $A(x)$) represent the value or the "grade of membership" of the x in A :

$$A = \{(x, \mu_A(x)) | x \in X\}.$$

Zadeh (1973) proposed another notation for fuzzy sets:

$$A = \sum_{x \in X} \mu_A(x)/x,$$

for countable universe X and:

$$A = \int_X \mu_A(x)/x,$$

for uncountable X . The \sum, \int signs denotes idempotent summation. For countable universe case this summation satisfies: $a/x + b/x = \max(a/b)/x$.

If the value of membership function is restricted to either 0 or 1 then A is reduced to a

classical set and $\mu_A(x)$ is the characteristic function $\chi_A(x)$. Because fuzzy sets correspond to the classical sets both fuzzy and classical sets have the corresponding basic operations of union, intersection and complement.

Let $F(X)$ denote a family (a class) of all fuzzy subsets of the universe of discourse X , i.e.

$$F(X) = \{\mu | \mu : X \rightarrow [0, 1]\}.$$

It can be easily proved that the laws of the excluded middle and contradiction (complementation) are not fulfilled. Therefore the structure $(F(X), \vee, \wedge,)$ forms the so called de Morgan (or soft) algebra rather than Boolean algebra.

The support of fuzzy set A is a crisp set that contains all elements with positive membership degree:

$$Supp(A) = \{x \in X | \mu_A(x) > 0\}.$$

Another crisp set connected with fuzzy set is the core of set A :

$$Core(A) = \{x \in X | \mu_A(x) = 1\}.$$

A more general notation is a α -level set (α -cut set) A_α . The α -level set is a crisp set that contains all elements with the membership greater than or equal to α , i.e.

$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\}.$$

By putting strong inequality, we obtain a strong α -level (strong α -cut) set. Hence the support is a strong 0-cut and the core is 1-cut sets.

A fuzzy set is convex if and only if:

$$\forall x, y \in X \forall \lambda \in [0, 1] \mu_A(\lambda x + (1 - \lambda)y) \geq \min(\mu_A(x), \mu_A(y)).$$

Fuzzy sets (A and B) are equal iff:

$$\forall x \in X \mu_A(x) = \mu_B(x).$$

Fuzzy set A is a subset from B ($A \subseteq B$) iff:

$$\forall x \in X \mu_A(x) \leq \mu_B(x).$$

A fuzzy set is completely characterized by its membership function. We define usually used classes of membership functions:

1. Triangular with a, b, c parameters:

$$\mu_A(x) = \begin{cases} 0, & x \leq a, \\ \frac{x-a}{b-a}, & a < x \leq b, \\ \frac{c-x}{c-b}, & b < x \leq c, \\ 0, & c \leq x \end{cases}$$

2. Trapezoidal with a, b, c, d parameters:

$$\mu_A(x) = \begin{cases} 0, & x \leq a, \\ \frac{x-a}{b-a}, & a < x \leq b, \\ 1, & b < x \leq c, \\ \frac{d-x}{d-c}, & c < x \leq d, \\ 0, & d \leq x \end{cases}$$

3. Gaussian with m and σ parameters:

$$\mu_A(x) = e^{-\frac{(x-m)^2}{2\sigma^2}}.$$

4. Fuzzy singleton with x_0 parameter:

$$\mu_A(x) = \begin{cases} 1, & x = x_0, \\ 0, & x \neq x_0 \end{cases}$$

A.1 Operation on fuzzy sets

Although the set-theoretic operations (union, intersection and complement) possess some rigorous axiomatic properties (see appendix B), they are not the only possible ones to interpret the respective connectives of fuzzy subsets of a given set X . There are also other reasonable and consistent operations on fuzzy sets representing disjunctions (unions) conjunctions (intersections) and negations (complements).

By the below outlined approach to conjunctions, disjunctions and strong negations we can define the intersection (\cap), the union (\cup) and the complement ($\bar{}$) of fuzzy subsets A, B of X as follows:

$$(A \cap_T B)(x) = T(A(x), B(x)) \quad \text{for all } x \in X,$$

$$(A \cup_S B)(x) = S(A(x), B(x)) \quad \text{for all } x \in X,$$

where $T(S)$ is any t-norm (s-norm), sometimes denoted as $\star_T(\star_S)$;

$$\overline{A(x)} = n(A(x)) \quad \text{for all } x \in X,$$

where n denotes a negation.

Ernest Czogała and Jacek Łeński ([13]) consider the class of intersection-union operators known as the triangular norms, i.e. t-norm T and t-conorm (s-norm) S operators considered as functions: $T : [0, 1]^2 \rightarrow [0, 1], S : [0, 1]^2 \rightarrow [0, 1]$. The T serves as a basis for defining intersections of fuzzy sets while S serves as a basis for defining unions of fuzzy sets. Taking into account the properties of classical sets the following axioms may be accepted:

1. Boundary conditions:

$$\begin{aligned} T(x, 1) = x, T(x, 0) = 0 & \quad \text{for all } x \in [0, 1], \\ S(x, 1) = x, S(x, 0) = 0 & \quad \text{for all } x \in [0, 1], \end{aligned}$$

2. Commutativity:

$$\begin{aligned} T(x, y) = T(y, x) & \quad \text{for all } x, y \in [0, 1], \\ S(x, y) = S(y, x) & \quad \text{for all } x, y \in [0, 1], \end{aligned}$$

3. Monotonicity:

If $x \leq u$ and $y \leq r$ then

$$\begin{aligned} T(x, y) &\leq T(u, y) & \text{for any } x, y, u \in [0, 1], \\ T(x, y) &\leq T(x, r) & \text{for any } x, y, r \in [0, 1], \\ S(x, y) &\leq S(u, y) & \text{for any } x, y, u \in [0, 1], \\ S(x, y) &\leq S(x, r) & \text{for any } x, y, r \in [0, 1], \end{aligned}$$

4. Associativity:

$$\begin{aligned} T(x, T(y, z)) &= T(T(x, y), z) & \text{for all } x, y, z \in [0, 1], \\ S(x, S(y, z)) &= S(S(x, y), z) & \text{for all } x, y, z \in [0, 1], \end{aligned}$$

More precisely a function $T : [0, 1]^2 \rightarrow [0, 1]$ is a triangular norm (t-norm) if and only if (iff) it satisfies the above written conditions 1. - 4. concerning T and a function $S : [0, 1]^2 \rightarrow [0, 1]$ is a t-conorm (s-norm) iff it satisfies conditions 1. - 4. concerning S . From the algebraic point of view T is a semigroup operation in $[0, 1]$ with identity 1.

Appendix B

Zermelo-Fraenkel set theory

Zermelo in 1908 was the first to attempt an axiomatisation of set theory. Many other mathematicians attempted to axiomatise set theory. Fraenkel, von Neumann, Bernays and Gödel are all important figures in this development.

The axioms for set theory now most often studied and used are called the Zermelo-Fraenkel set theory (ZF). Actually, this term usually excludes the axiom of choice. When this axiom is included, the resulting system is called ZFC.

An important feature of ZFC is that every object that it deals with is a set. In particular, every element of a set is itself a set. Other familiar mathematical objects, such as numbers, must be subsequently defined in terms of sets.

The ZFC axioms are:

- Extensionality: Two sets are the same if and only if they have the same members.

$$\forall A, \forall B : A = B \iff (\forall C : C \in A \iff C \in B)$$

- Empty set: There exists a set with no members, called the empty set and denoted \emptyset . A redundant axiom.

$$\exists \emptyset, \forall A : \neg(A \in \emptyset)$$

- Pairing: If x, y are sets, then there exists a set, denoted $\{x, y\}$, whose sole members are x and y . Replacement makes this redundant.

$$\forall A, \forall B, \exists C, \forall D : D \in C \iff (D = A \vee D = B)$$

- Union: For any set x , there is a set y such that the elements of y are precisely the members of the members of x .

$$\forall A, \exists B, \forall C : C \in B \iff (\exists D : C \in D \wedge D \in A)$$

- Infinity: There exists a set x such that \emptyset is a member of x , and whenever y is in x , so is $y \cup \{y\}$.

$$\exists \mathbf{N} : \emptyset \in \mathbf{N} \wedge (\forall A : A \in \mathbf{N} \implies A \cup \{A\} \in \mathbf{N})$$

- Power set: Given any set, its power set exists. That is, for any set x there exists a set y , such that the members of y are precisely the subsets of x .

$$\forall A, \exists \mathcal{P}A, \forall B : B \in \mathcal{P}A \iff (\forall C : C \in B \implies C \in A)$$

- Regularity: Every non-empty set x contains some member y such that x and y are disjoint sets.

$$\forall A : A \neq \emptyset \implies \exists B : B \in A \wedge \neg \exists C : C \in A \wedge C \in B$$

- Separation (or subset axiom): Given any set and any proposition $P(x)$, there exists a subset of the original set containing precisely those members x for which $P(x)$ holds. (This is an axiom schema.) Replacement makes this redundant.

$$\forall A, \exists B, \forall C : C \in B \iff C \in A \wedge P(C)$$

- Replacement: Given any set A and any functional mapping, defined as a dyadic relation $P(x, y)$ such that $P(x, y_1)$ and $P(x, y_2)$ implies $y_1 = y_2$, there is a set containing precisely the images of the members of A . Colloquially, if the domain of a function is a set, its range is as well. (This is an axiom schema.)

$$(\forall X, \exists ! Y : P(X, Y)) \rightarrow \forall A, \exists B, \forall C : C \in B \iff \exists D : D \in A \wedge P(D, C)$$

- Choice: Given any set of pairwise disjoint non-empty sets, there exists a set that contains exactly one member from each of these non-empty sets.

$$\begin{aligned} \forall A : & ((\forall B : B \in A \rightarrow (\exists C : C \in B \wedge \forall D : (D \in A \wedge D \neq B \rightarrow \neg \exists E : E \in B \wedge E \in D))) \\ & \rightarrow \exists F, \forall G : (G \in A \rightarrow \exists ! H : H \in G \wedge H \in F)) \end{aligned}$$

Appendix C

Definitions

Definition C.0.1: A partial order on a set A is a relation $\leq \subset A \times A$ that satisfies the following three properties:

- Reflexivity: $x \leq x$ for all $x \in A$
- Antisymmetry: If $x \leq y$ and $y \leq x$ for any $x, y \in A$, then $x = y$
- Transitivity: If $x \leq y$ and $y \leq z$ for any $x, y, z \in A$, then $x \leq z$

Definition C.0.2: A relation $\leq \subset A \times A$ is a total order, linear order or simple order on a set A if the following properties hold:

- Antisymmetry: If $x \leq y$ and $y \leq x$ for any $x, y \in A$, then $x = y$
- Transitivity: If $x \leq y$ and $y \leq z$ for any $x, y, z \in A$, then $x \leq z$
- Comparability/ totality: for any $x, y \in A$, either $x \leq y$ or $y \leq x$

A set paired with an associated total order on it is called a **totally ordered set**, a linearly ordered set, a simply ordered set, or a chain.

Notice that the totalness condition implies reflexivity, that is $a \leq a$. Thus a total order is also a partial order. A total order can also be defined as a partial order that is total.

Definition C.0.3: A well-order (or well-ordering) on a set A is an order relation on A with the property that every non-empty subset of A has a least element in this ordering. The set A together with the well-order is then called a well-ordered set. A well-order is necessarily a total order.

Definition C.0.4: Let \mathcal{F} be a collection of subsets of a sample space Ω . \mathcal{F} is called a σ -field (or σ -algebra) if and only if it has the following properties:

1. The empty set $\emptyset \in \mathcal{F}$.
2. If $A \in \mathcal{F}$, then the complement $A^c \in \mathcal{F}$.

3. If $A_j \in \mathcal{F}$, $j = 1, 2, \dots$, then their union $\bigcup A_j \in \mathcal{F}$.

A pair consistin of a set Ω and a σ -field \mathcal{F} of subsets of Ω is called a *measurable space*. The elements of \mathcal{F} are called measurable sets in measure theory or *events* in probability and statistics.

Definition C.0.5: Let (Ω, \mathcal{F}) be a measurable space. A set function ν defined on \mathcal{F} is called a **measure** if and only if it has te following properties.

- $0 \leq \nu(A) \leq \infty$ for any $A \in \mathcal{F}$.
- $\nu(\emptyset) = 0$.
- If $A_j \in \mathcal{F}$, $j = 1, 2, \dots$ and A_j 's are mutually disjoint, i.e., $A_j \cap A_k = \emptyset$ for any $j \neq k$, then

$$\nu\left(\bigcup_{j=1}^{\infty} A_j\right) = \sum_{j=1}^{\infty} \nu(A_j)$$

The triple $(\Omega, \mathcal{F}, \nu)$ is called a *measure space*. If $\nu(\Omega) = 1$, then ν is called a *probability measure* and $(\Omega, \mathcal{F}, \nu)$ is called a *probability space*.

Bibliography

- [1] Xiaomeng Wang and C. Borgelt. Information measures in fuzzy decision trees. In *Fuzzy Systems, 2004. Proceedings. 2004 IEEE International Conference*, volume 1. IEEE Press.
- [2] J. F. Baldwin, J. Lawry, and T. P. Martin. Mass assignment fuzzy id3 with applications. In *the proceedings of Fuzzy Logic: Applications and Future Directions Workshop, London, UK*, pages 278–294, 1997.
- [3] J. F. Baldwin and Dong Xie. Simple fuzzy logic rules based on fuzzy decision tree for classification and prediction problem. pages 175–184, 2005.
- [4] H. Bass, J. F. C. Kingman, F. Smithies, J. A. Todd, and C. T. C. Wall. *Algebraic graph theory*. Brooke Crutchley, Cambridge, 1974.
- [5] Hyman Bass and Alexander Lubotzky. *Tree Lattices (Progress in mathematics; v.176)*. Birkhäuser-Verlag, Boston, 2001.
- [6] Benno Biewer. *Fuzzy-Methoden*. Springer-Verlag, Berlin, Heidelberg, 1997.
- [7] Xavier Boyen and Louis Wehenkel. Automatic induction of fuzzy decision trees and its application to power system security assessment. *Fuzzy Sets and Systems*, 102:3–19, 1999.
- [8] Leo Breiman. Bagging predictors. technical report. Technical Report 421, University of California, 1994.
- [9] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [10] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
- [11] Ahlame Chouakria-Douzal and Panduranga Nagabhushan. Improved fréchet distance for time series. In V. Batagelj, H.-H. Bock, A. Ferligoj, and A. Žiberna, editors, *Data Science and Classification*. Springer Berlin Heidelberg, 2006.
- [12] Dietmar Cieslik, Andreas Dress, and Walter Fitsch. *The Double Tree Algorithm*. Universität Bielefeld, 1999.

- [13] Ernest Czogała and Jacek Łeński. *Fuzzy and Neuro-Fuzzy Intelligent Systems*. Physica-Verlag, Heidelberg, 2000.
- [14] Reinhard Diestel. *Graph Theory*. Springer, New York, 2000.
- [15] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.
- [16] A. Dress, D.Huson, and V. Moulton. *Analyzing and Visualizing Sequence and Distance Data Using Splitstree*. Universität Bielefeld Forschungsdchwerpunkt Mathematisierung- Strukturbildungsprozesse, Biefeld, 1995.
- [17] Andreas Dress and Werner Terhalle. *The Real Tree*. Universität Bielefeld, 1995.
- [18] Heinz W. Engl, Martin Hanke, and Andreas Neubauer. *Regularization of Inverse Problems*. Kluwer Academic Publishers, Dordrecht, 2000.
- [19] Jeff Erickson. Lower bounds for external algebraic decision trees. *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*.
- [20] Jeff Erickson. Algebraic decision trees in concrete models of computation. Concrete Models of Computation CS 497 JE, 2003.
- [21] Jeff Erickson. Lower bounds for selection. Concrete Models of Computation CS 497 JE, 2003.
- [22] Ludwig Fahrmeir, Alfred Hamerle, and Gerhard Tutz. *Multivariate statistische Verfahren*. Walter de Gruyter, 1996.
- [23] Charles Favre and Mattias Jonsson. *The Valuative Tree*. Springer-Verlag, Berlin, Heidelberg, 2004.
- [24] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: an overview. pages 1–34, 1996.
- [25] William Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley and Sons, New York, Chichester, Brisbane, Toronto, 1968.
- [26] Fernando Berzal Galiano. *ART: Un método alternaivo para la construcción de árboles de decisión*. PhD thesis, Universidad de Granada, 2002. In spanish.
- [27] A. D. Gordon. *Classification*. Chapman and Hall, London, 1981.
- [28] Ewa Graczyńska. *Universal Algebra via tree operads*. Opole, 2000.
- [29] Jiawei Han and Micheline Kamber. *Data Mining. Concepts and Techniques*. Academic Press, London, 2001.

- [30] Jan Hauth. Grey-box modelling for nonlinear systems. 2007.
- [31] Symon Haykin. *Neural Networks. A comprehensive Foundation*. Prentice-Hall, New Jersey, 1994.
- [32] Tu Bao Ho, David Cheung, and Huan Liu. *Advances in Knowledge Discovery and Data Mining: 9th Pacific-Asia Conference, PAKDD*. Springer, Vietnam, 2005.
- [33] Jane Yung-jen Hsu I-Jen Chiang. Fuzzy classification trees for data analysis. *Fuzzy sets and systems*, 130(3):87–99, 2001.
- [34] A. C. Yao J. M. Steele. Lower bounds for algebraic decision trees. *J. Algorithms*, 3:1–8, 1982.
- [35] Cezary Z. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 28:1–14, 1998.
- [36] R. Kruse and K.D. Meyer. *Statistics with Vague Data. Theory and Decision Library*. D. Reidel Publishing Company, 1987.
- [37] Mark Last, Oded Maimon, and Einat Minkov. Improving stability of decision trees. *International journal of pattern recognition and artificial intelligence*, 16:145–159, 2002.
- [38] Ruey-Hsia Li. *Instability of decision tree classification algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.
- [39] Fabio Maccheroni, Massimo Marianacci, and Aldo Rustichini. A variational formula for the relative gini concentration index. *ICER Working Paper*, 5/2004, 2004.
- [40] Ralf Mikut, Jens Jäkel, and Lutz Groll. Interpretability issues in data-based learning of fuzzy systems. *Fuzzy sets and systems*, 150:179–197, 2005.
- [41] H.D. Mittelman and H. Weber. *Bifurcation Problems and their Numerical Solution - Workshop on Bifurcation Problems and their Numerical Solution*. Birkhäuser-Verlag, Dortmund, 1980.
- [42] M. Mlynski, U. Bucken, K. Markus, R. Koos, V. Lang, W. Ameling, and P. Hanrath. Using scalar fuzzy control for aggregation problems in knowledge based diagnosis for implantable devices. *Engineering in Medicine and Biology Society, 2003. Proceedings of the 25th Annual International Conference of the IEEE*, 2003.
- [43] Michael F. Mlynski. *Eine neue Methode des unscharfen Schließens für Expertensysteme*. PhD thesis, University of Aachen, 2003.
- [44] Shlomo Moran, Marc Snir, and Udi Manber. Applications of ramsey’s theorem to decision tree complexity. *J. ACM*, 32(4):938–949, 1985.

- [45] Heiko Müller, Johann Christoph Freytag, and Ulf Leser. Describing differences between databases. In *CIKM*, pages 612–621, 2006.
- [46] Detlef D. Nauck. *Data analysis with Neuro-Fuzzy Methods*. PhD thesis, University of Magdeburg, 2000.
- [47] Cristina Olaru and Louis Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138:221–254, 2003.
- [48] Petra Perner. *Data Mining on Multimedia Data*. Springer-Verlag, Berlin, Heidelberg, New York, 1998.
- [49] Petra Perner. Improving the accuracy of decision tree induction by feature pre-selection. *Applied artificial intelligence*, 15:747–760, 2001.
- [50] Scott M. Pike. Binary trees: A challenge problem for separating concerns. in *Proceedings of the ICSE 2001 Workshop on Advanced Separation of Concerns in Software Engineering*.
- [51] Hans Jürgen Prömel and Angelika Steger. *The Steiner Tree Problem*. Vieweg, Braunschweig/Wiesbaden, 2002.
- [52] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [53] Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41:77–93, 2004.
- [54] Beatriz Clavero Rasero. *Statistical aspects of setting up a credit rating system*. PhD thesis, University of Kaiserslautern, 2006.
- [55] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.
- [56] Gunter Rote. Computing the minimum hausdorff distance between two point sets on a line under translation. *Information Processing Letters*, 38(3):123–127, 1991.
- [57] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. *Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India*, pages 59–66, 1998.
- [58] Walter Rudin. *Principles of Mathematical Analysis*. Mc Graw-Hill, 1976.
- [59] Enrique H. Ruspini. A new approach to clustering. *Information and Control*, 15(1):22–32, 1969.
- [60] Steven Salzberg. *C4.5: Programs for Machine Learning by J. Ross Quinlan*, volume 16. Kluwer Academic Publishers, Boston, 1994.

- [61] Jun Shao. *Mathematical Statistics*. Springer-Verlag, New York, 2003.
- [62] S. S. Shapiro and M. B. Wilk. *An analysis of variance test for normality (complete samples)*. Oxford Journals, 1965.
- [63] King-Sun Fu Shi Qing-Yun. A method for the design of binary tree classifiers. *Pattern Recognition*, 16.
- [64] Peter J. Tanand and David L. Dowe. Mml inference of oblique decision trees. *The 17th Australian Joint Conference on Artificial Intelligence*, 2004.
- [65] Kati Viikki. *Machine Learning on Ontoneurological Data: Decision Trees for Vertigo Diseases*. University of Tampere, Finland, 2002.
- [66] Leland Wilkinson. Tree structured data analysis: Aid, chaid and cart. 1992.
- [67] King-Sun Fu Y. K. Lin. Automatic classification of cervical cells using a binary tree classifier. *Pattern Recognition*, 16(1):69–80, 1983.
- [68] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69:125–139, 1995.
- [69] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.
- [70] E. Zintzaras, N.P. Brown, and A. Kowald. Growing a classification tree using the apparent misclassification rate. *Laboratory of Mathematical Biology. National Institute for Medical Research.*, 10:263–273, 1994.